AN INVESTIGATION OF PROGRAM EXCHANGE METHODS FOR A MUTIPROGRAMMING ENVIRONMENT

ROSS R. HATCH

M11

AN INVESTIGATION OF PROGRAM EXCHANGE METHODS FOR A MULTIPROGRAMMING ENVIRONMENT

* * * * *

Ross R. Hatch

AN INVESTIGATION OF PROGRAM EXCHANGE METHODS FOR A MULTIPROGRAMMING ENVIRONMENT

by

Ross R. Hatch

Lieutenant, United States Navy

Submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE
IN
ENGINEERING ELECTRONICS

United States Naval Postgraduate School
Monterey, California

1 9 6 4

AN INVESTIGATION OF PROGRAM EXCHANGE METHODS FOR A MULTIPROGRAMMING ENVIRONMENT

by

Ross R. Hatch

This work is accepted as fulfilling

the thesis requirements for the degree of

MASTER OF SCIENCE

IN

ENGINEERING ELECTRONICS

from the

United States Naval Postgraduate School

ABSTRACT

An investigation of program exchange techniques and methods of evaluating such procedures is conducted. Basic hardware and system parameters vitally affecting program exchange are discussed. The basic program exchange methods covered are 1) the Complete Program Exchange 2) the Block or Page Exchange and 3) the Completely Integrated System Exchange. The investigation is conducted using heuristic analysis, and a simulation study is conducted on selected methods. The combined analysis not only evaluates present methods but provides a guide for evaluating and selecting an Exchange technique for any system configuration. A complete multiprogramming system simulator and a specific technique for status preservation are presented as Appendices.

The author wishes to express his appreciation to Mr. Jules I. Schwartz and the members of the ARPA Time-Sharing Project, System Development Corporation, Santa Monica, California for their assistance and to Professor Mitchell L. Cotton for his invaluable advice and encouragement during this investigation.

TABLE OF CONTENTS

APPENDICES

LIST OF ILLUSTRATIONS

TABLE OF SYMBOLS AND ABBREVIATIONS

CDC	Control Data Corporation
SDC	System Development Corporation
TSS	Time-Sharing System
ECR	Executive Control Routine
I/O	Input/Output
K	1000 memory units
N_{max}	Maximum number of operating user stations permitted in the system
n_{max}	Maximum number of users that can receive service during a given cycle
t_r	System response or cycle time
t_s	Exchange time
q	Quantum
η	System Executive overhead expressed as a percentage of a full cycle

1. Introduction

Recent technological advances in high speed logic, digital data transmission, random access mass storage devices and related fields have freed the system designer from many former constraints. The resulting complex large scale systems will be able to operate in a practical and efficient manner only through the use of a technique such as multiprogramming. The term, multiprogramming, is applicable primarily to the computer with a single processing unit and is defined as the execution of several programs by the transferring of control among them in a controlled fashion, but where one and only one program is in control at any one time. (9)

Multiprogramming, by definition, includes the concept of providing simultaneous service to several on-line users. However, the majority of the work in the field is involved with concurrent type operations; the interleaving of various type programs to improve overall efficiency. A common example of this is the combination of the compute limited and the I/0 limited program to permit each element of the system (i.e. the central processor and each peripheral device) to use a greater portion of the operating cycle. The goal of this type of operation is the fulltime use of every system element.

An area which is now receiving attention, and deserves much more, is that of time-sharing, or the furnishing of service to several on-line users. Bright and Chedleur have suggested the term "multiple break in operation" to more aptly describe this type of operation which

concentrates on user service. (3) Due to inherent reaction time delays in man-machine communications, several on-line users can obtain virtually simultaneous service. The system can offer assistance simultaneously to several programmers for on-line debugging and program modification in the program formulation phase; provide a control system for war game simulation and data retrieval; and perform various command and control functions. Short bursts of service are thus provided to several on-line users, while the batch jobs, serving as a system background, are only slightly degraded. This paper will approach multiprogramming in this service context. Accordingly, time-sharing, which has been used in various ways in the literature, will be defined as the essentially simultaneous use of a central processor by several on-line users.

The introduction of multiple simultaneous users into a system immediately creates severe control problems. A comprehensive Executive Control Routine (ECR) is required to exercise positive control over the system. The characteristics of this routine and effectiveness of its control will exert a dominant influence on the overall system efficiency.

One of the most difficult problems faced by the Executive Control Routine is program exchange. To permit several simultaneous users in the system, an efficient method must be devised to exchange programs while retaining all status information and program environment. This information must be readily available for use when restoring a previously terminated program. As control is passed from one program to another, the loading cycle must include the resetting of the environment and

status. This paper will investigate and analyze the various aspects of the Exchange problem and outline the general hardware and software requirements. Particular characteristics will be noted and a general analytic procedure to evaluate various exchange methods developed. A System Simulator will be used where applicable to test the effectiveness of various exchange techniques on overall system performance. The final result will not only furnish a summary of exchange methods available, but provide a general analytic technique using both heuristic and simulation methods to evaluate future exchange methods.

2. Background

Several widely diverse interest groups can benefit from a time-sharing system. Each, individually, would impose slightly differing timing and control problems, but these could be accommodated relatively easily in a general system. The technical knowledge of each group will vary widely, and the system must retain the capability of serving the most experienced and well trained user while providing service to the neophyte. The following paragraphs will describe several areas where time-sharing could be utilized to good advantage. Some are current problems that are presently handled by other means, while others are uses that would become economically feasible through the use of an approach such as time-sharing.

The programmer's debugging problem is too time consuming to permit individual use of a large computer. Time-sharing, however, provides virtually instantaneous computer reaction time and allows others to operate while the programmer is interpreting results. The excessive turn-around time experienced in closed shop batch job operation is thus avoided, or at least, considerably reduced. Another desirable feature oriented toward the engineer/programmer is the ability to make repeated runs, changing parameters on the basis of preceding runs without closed shop delays. This is easily accomplished under time-sharing.

Real time operations fall into the purview of the time-sharing system. In control and monitoring applications, where the demands

for service are normally absolute, the system can easily become saturated, and extreme care must be taken when including this type of activity in an operating system.

On-line data retrieval appears to be the most promising of the future applications of time-sharing. A user will have access, from a remote station, to a large data base. This will not only allow faster handling of tasks that were formerly done manually, but encourage the use of data retrieval to enable more informed decisions to be made. The ability to provide such a service to a multitude of users makes it economically feasible for all. Banking uses and airline reservation systems fall into this general system.

3. The Executive Control Routine

All multiprogramming and multiprocessing systems from the basic stack job processor to the most complex on-line system depend on an Executive Control Routine for their effectiveness. The increasing interest in multiprogramming systems has been reflected in the greater emphasis being placed on the general subject of control routines The primary requirements of the Executive Control Routine are reliability and effectiveness. The user must be able to assume that the system program is error free and virtually fool proof, and that long periods of service can be expected. This implies that the Executive has the capability of recovering from both object program and machine errors.

Though terminology may differ, the Executive Control Routine consists of five basic parts. The Interrupt Handler passes control to various parts of the Executive and establishes the initial flow. The Scheduler determines what programs are to be run, in what order, and for how long. The Dispatcher handles all I/0 transfers, and the Sequencer establishes the normal flow through the entire Executive. The Exchange Routine uses the information from the Scheduler as inputs to allocate internal and external storage and initiate program transfers through the Dispatcher. Other functions of the Executive include Rollback and Recovery, interpretive packages and, if provided, debugging routines. The Executive must, in general terms, perform as an efficient and capable executive or supervisor, and the broad requirements can be determined without difficulty. It is when specific portions are subjected

to detailed investigation that the difficulties become more apparent. The constraints and problems created by a specific system or approach will vitally affect the system in question, and all facets must be carefully considered.

To provide a general idea of how the Executive functions, the sequence of operation of a typical time-sharing system will be described. All object programs are initially stored in a high speed random access store, placed there by a load command to the Program Exchange Routine. At the start of a basic cycle, the Scheduler determines the queue, and the Exchange Routine brings the required program into core at, or preferably before, its actual active quantum. If a storage conflict occurs, the previous programs are transferred to the external store as required. At the completion of a program's active period, whether through an early termination or the normal end of a quantum, the program environment (i.e. statue of all operational registers, etc.) is saved and, dependent upon the system load, the program is either saved in core or transferred to the external store awaiting its next turn. A basic concept that will be followed throughout this paper is that no user will be transferred from core unless a storage conflict exists. The specific conflicts will be determined by the exchange method as will procedures to reduce both the probability and the effect of such conflicts.

This paper is primarily concerned with the Program Exchange portion of the Executive Routine, and it will be assumed that the remainder of the Executive performs its basic functions in a normal

manner. If any deviations are required by a particular exchange method, they will be noted.

4. Program Exchange

As long as the compute speeds greatly exceed I/0 rates, as they do at present, the program exchange phase will remain a critical operation. When the Scheduler determines a request is to be honored, the Exchange Routine determines what jobs, if any, need to be dumped to provide the required core space, where to load the new program, and saves all required information, sets memory protection limits when applicable and handles relocation if provided in the system. The Exchanger also handles space allocation and maintenance in both core and in external storage devices.

4.1 General Considerations

The effectiveness of the exchange technique will be determined to a great extent by the hardware configuration of the system. Memory protection is of vital importance, and it should be emphasized that without this feature the integrity of even the Executive Routine cannot be guaranteed. Relocatibility is both a hardware and a software feature. Lack of this capability seriously hampers the system in that dynamic space allocation is virtually impossible and space is essentially allocated by the compiler.

Although several of the exchange algorithms to be discussed are hardware limited, some can be used to overcome hardware deficiencies and improve the system. The system itself determines to a great extent the type of swapping used, and core utilization must be carefully weighed against overhead time required. In the analysis of the exchange algorithms,

the emphasis will be placed on hardware requirements, overhead and core utilization with the aim of providing the user with the most efficient service. The basic approaches will be handled first, and then various combinations investigated to find the most effective methods of solving the swap problem.

The normal exchange methods can be divided into two basic groups, those that swap entire programs, and those that handle blocks or "pages", the blocks or "pages" being defined as data blocks less than program size in length. Each method has its particular advantages and, although complete program swaps have been favored in the past, the increase in the number of users in a time-sharing system coupled with the reaction time required and the increasing size of programs calls for a critical re-evaluation of both methods. Many of the problems are common to several techniques and will be discussed in detail when they first occur and only mentioned in subsequent methods.

4.2 Exchange Techniques and System Capacity

The reaction time of the system is one of the basic parameters of a time-sharing system. The decision as to reaction time will vitally effect both the user and the system. A selection of too small a reaction time (t_r) will seriously decrease the number of active users serviced during a cycle while too long a period will result in disgruntled users. Although the users are receiving better service than could be expected in a closed shop operation, personal observation has shown that the addition of even a few seconds delay will tend to cause general

discontent among users. A second basic system parameter is the quantum time or the time alloted to a specific user during a given cycle. The determination of quanta is treated in detail by Lt. W. G. Wilder (31). The Executive Control Routine establishes an operating queue and passes control to the proper programs. The only delay in the passing of control is the non-availability of the program in core. Due to the length of transfer times relative to compute times, the basic exchange time (t_s) will normally exceed the quantum and will become the limiting factor in the number of users serviced. In the normal passing of control, the program in core is dumped, the required program loaded and run for the quantum. Thus ($2t_s + q$) is required for each user and the maximum number of active users per cycle is given by:

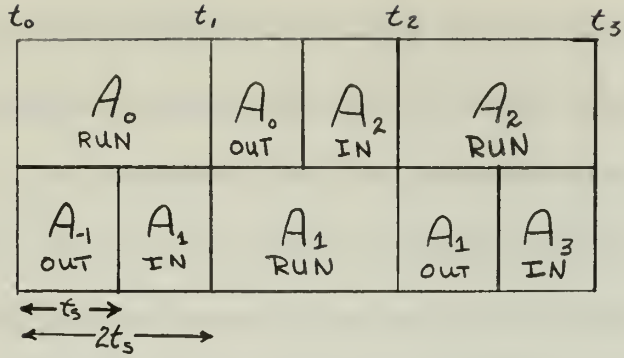
$$n_{max} = t_r(1-\eta)/ (2t_s+q)$$

This is based on the complete swap approach when, due to hardware limitations and/or program size, only one program can exist in the core at any one time. These constraints will be discussed more fully in the following sections.

If programs are relocatable, the two users may exist in core

$$n_{max}=t_r(1-\eta) / 2t_s$$

and q may be as great as $2t_s$ with no degradation of service. The process is illustrated in Fig. 1 which starts at an arbitrary instant with program A_o running.



Holds for $q \geq 2t_s$

t_s = time to transfer half of core

A BASIC SPACE SHARING SCHEME
FIGURE 1

It would appear that n_{max} can be further increased by allowing more users in core and more simultaneous transfers, and this is true However, this requires multiple high speed I/0 channels, and this, in itself, presents new problems.

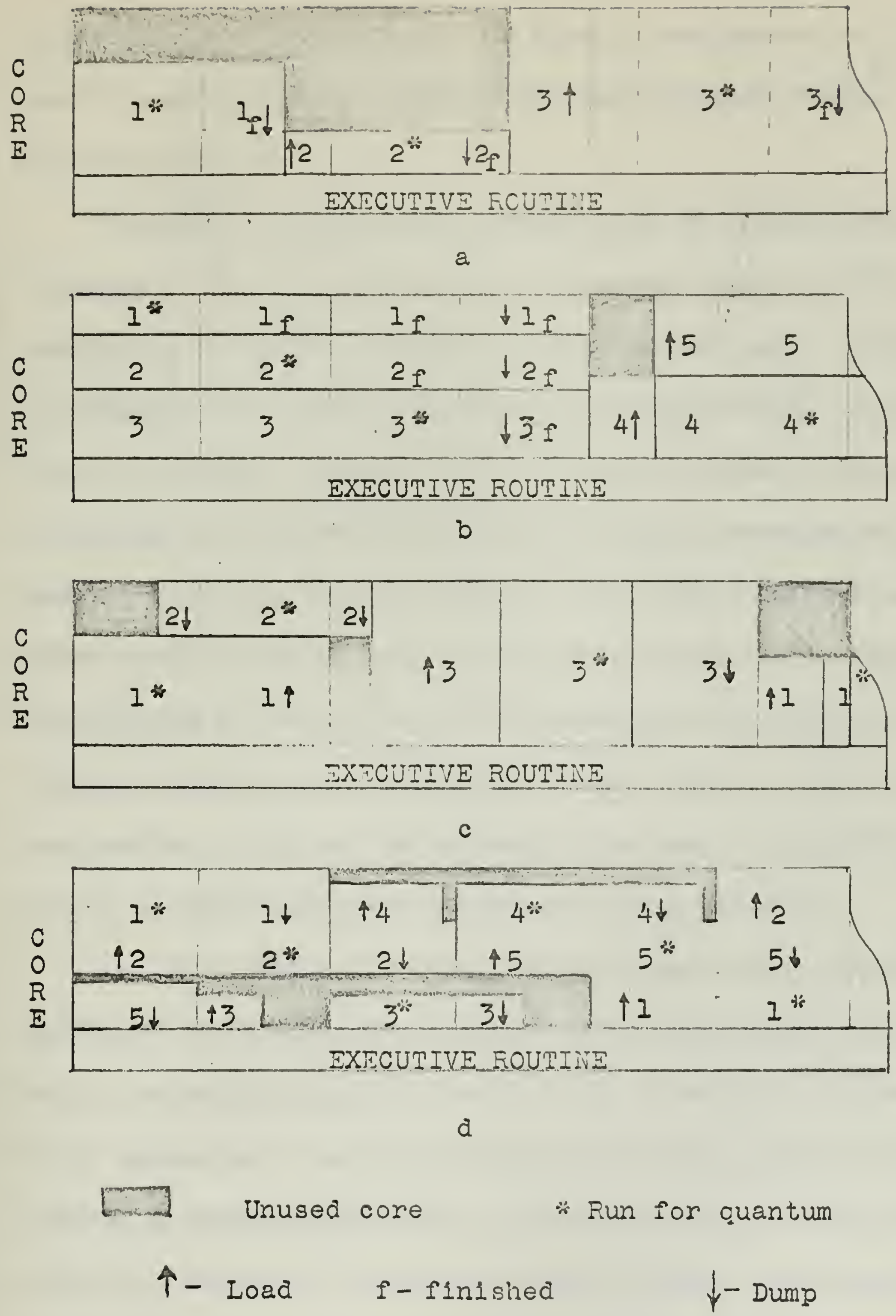
The analysis of particular methods of program exchange is intended to both delineate the capabilities and limitations of the individual procedures and also to demonstrate a general approach to the exchange problem. As new hardware developments permit new techniques for program exchange, these can be analyzed in a similar manner and evaluated against the same standards. Before treating individual methods of exchange, a discussion of generalized techniques such as space allocation, memory protection, relocation and random access storage mediums will provide a general background and permit certain vital hardware capabilities to be noted without repeated development.

4.3 Space Allocation

The preceding discussion of the effect of exchange methods on system performance also raised the problem of space sharing or space allocation. The examples used, considered extreme cases, whereas this section will treat the general allocation problem. Space allocation methods and program exchange techniques are closely interwoven, and the complete specification of one virtually determines the other. Because of this interdependence, space allocation is of vital importance in both the complete program and the block exchange concepts. The allocation scheme may range in complexity from the basic, one user approach, to full packing and even to the interleaving of instructions found in languages such as LISP and IPL. The basic ideas presented will be applied in various exchange methods, and the specific analysis will show the details involved and enumerate the hardware requirements that are imposed.

The graphic representation if Fig. 2 will be used to illustrate the dynamic changes in core storage during a typical interval of each of the basic allocation types. Due to the multiprogramming emphasis intended, it will be assumed that the Executive remains in core, and the allocation methods will be concerned only with the remaining core.

The basic single user allocation depicted in Fig. 2a. This is the type normally used in a system control program such as the Fortran Monitor and results in low core utilization factors, typically 0.10. The space allocation in this case consists of simply assigning all



BASIC SPACE ALLOCATION METHODS

FIGURE 2

available core to the next user. The effect of this approach will be more fully explored in the analysis of the basic complete program exchange method.

The dense, complex packing shown in Fig. 2b is used to best advantage in the system where memory is initially loaded, and then all programs run to completion before any exchanges are made. This type of multiprogramming system, typified by the Ferranti FP6000, concentrates on utilization efficiency and is not concerned with on-line users. The complexity of packing for each load can lead to complications and delays if the load is frequently changed. This method does not seem suited to any great extent to the time-sharing system with its constantly changing load and the requirements for exchanges every quantum. The problems created by time-sharing can be seen in Fig. 2c, where, although some packing is attained, the overhead for the space sharing allocation plus the occasional dead time reduces the system efficiency.

Fig. 2d is similar to the ideal block allocation case previously mentioned. The average core utilization factor lies between case a and b, and, as the programs approach the block size in length, the higher factor of b is approached. The method requires either large memories, short programs or program modification, but avoids many of the disadvantages of the other methods. For maximum system efficiency, the exchange time, t_s, should be completely overlapped by the minimum q. This requires that for Fig. 1, q=2ts, while Fig. 2d requires a $q=t_s$; the methods do, however, require one and two high speed transfer channels

respectively. The constraints imposed by this requirement will be discussed in the development of the complete program exchange.

4.4 External Storage Devices

When a program exchange is made, the program being removed must, obviously, be placed in some external storage device. The types of external storage available to the system will vitally affect the exchange method selected and the efficiency of the exchange phase of the control routine. Transfer rates and access times vary greatly not only between types, but among classes of specific types. A brief description of various external storage devices will establish a background for future decisions and delineate possible problem areas.

The ideal storage device is the magnetic core memory which provides an extremely high speed, random access storage device. Cost factors, however, render this impractical for the storage of large amounts of data. The widely varying loads to be expected in a multiple on-line user environment require the capability of storing the programs of all active users. The amount of external storage available for program exchange will establish one of the basic limits on the number of users allowed into the system at a given time. This limit is the maximum number of stations in use, and not the lower valued number of active users in a given cycle. While disc and drum storage, both random access devices, are the most applicable to the multiprogramming exchange problem, the magnetic tape system is included, due both to its flexibility and its low cost. These, along with some of the less common

storage devices, will be described in the following paragraphs. A summary of representative storage devices is presented in the graph of Figure 3 and the Table of Figure 4.

4.4.1. Magnetic drums

Magnetic drums provide an extremely fast random access capability at a lower cost than core. The magnetic tracks on the circumference of the drum may be read by either fixed or movable read/write heads. The circular track implies a maximum rotational delay of one drum rotation and an average time of half this amount. The movable head drum requires proportionally fewer heads and has a lower cost per character than the fixed head type. Positioning time, however, is increased from zero in the fixed head system to typical values of 50 to 300 msec. for various movable head systems. The cost vs. access time relationship can more easily be seen in Fig. 3 and Fig. 4.

4.4.2 Magnetic discs

Discs files are a natural step from the drum approach. Several discs mounted on a common shaft provide a lower cost per character than found in the drum devices. The time required to position mechanically the read/write heads plus an increased rotational delay makes the typical disc considerably slower than the drum. Various schemes for positioning the access arms are used, but all must solve difficult, but basically mechanical, problems. Of major importance among these problems are increasing the speed and accuracy of positioning and prevention of damage to the disc surface due to physical contact with

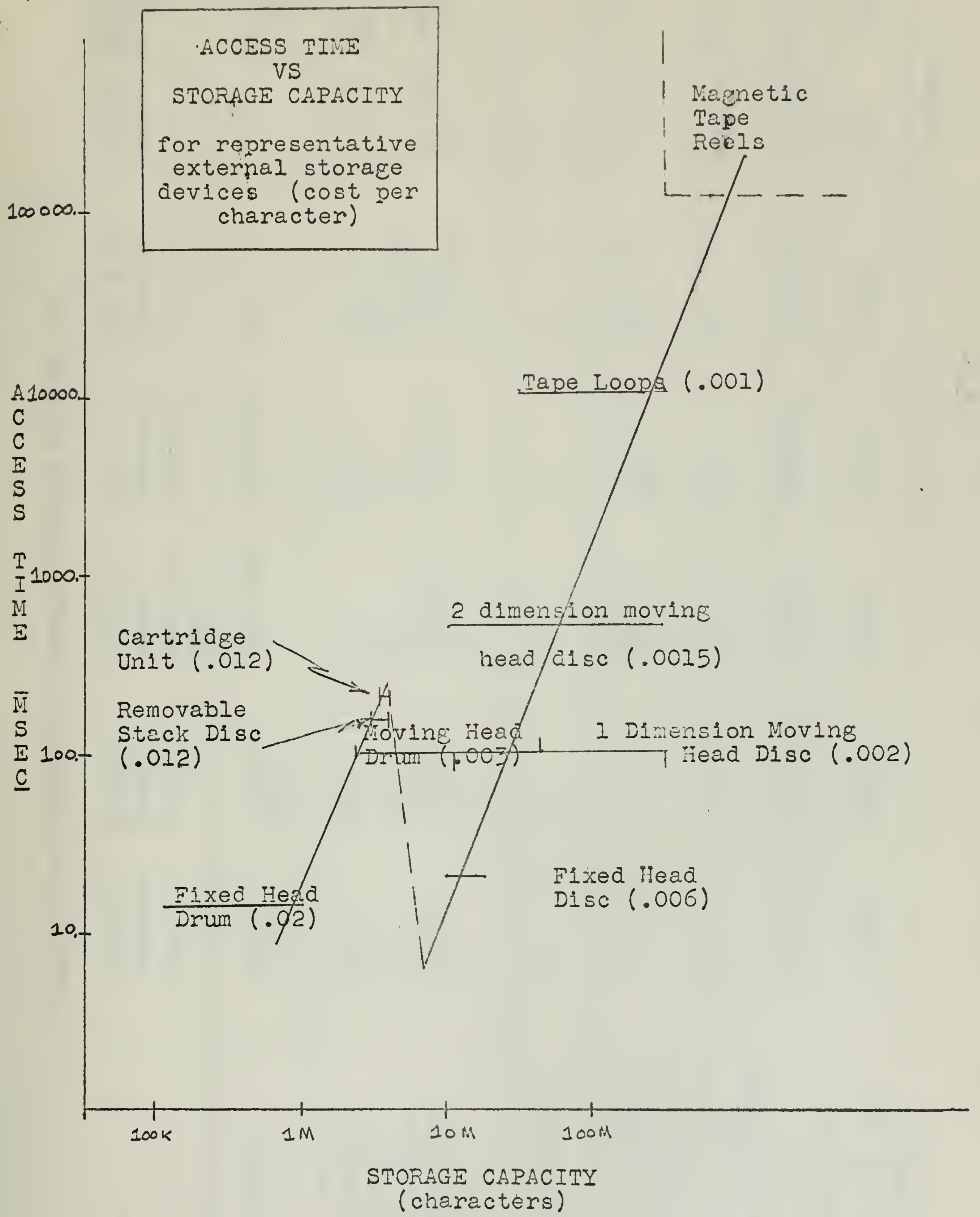

ACCESS TIME
VS
STORAGE CAPACITY
for representative external storage devices (cost per character)
Magnetic Tape Reels
100000.
A 10000.
ACCESS TIME MSEC
1000.
100.
10.
Tape Loops (.001)
2 dimension moving head disc (.0015)
Cartridge Unit (.012)
Removable Stack Disc (.012)
Moving Head Drum (.003)
1 Dimension Moving Head Disc (.002)
Fixed Head Disc (.006)
Fixed Head Drum (.02)
100K
1M
10M
100M
STORAGE CAPACITY (characters)

FIGURE 3

TABLE OF REPRESENTATIVE EXTERNAL STORAGE DEVICES /28/

Device	Univac Fastrand Drum	Univac FH-880 Drum	Data Pro-ducts-CDC 818 Disc	Data Products TSS Disc	NCR 353-1 Cram Unit	Woven Screen	Burroughs B 472
Typical System	Univac 1107	Univac 1107	CDC 1604A	TSS(Q-32)	NCR 315	-----	Burroughs B 5000
Storage Medium	drum	drum	disc	disc	magnetic	screens	disc
medium/unit	2	1	16	16	cards	-----	20
tracks/medium	3072	128	256	64	256 cards	-----	50
characters/unit	66Meg	4.7 Meg	33.5Meg	33.5Meg	7/card	10 Meg	48Meg
max units	120	120	28	--	5.55Meg	-----	20
					16		
Removable storage Medium	no	no	no	no	yes	no	no
Head Positioning (min, average, max) milliseconds	30,57,86	0,0,0	35, 120, 200	35, 125, 225	235, 235 235	10 sec	0,0,0
Rotational Delay (average, mseconds)	35	16.7	26	25	23	0	20
Transfer Rate Characters/sec	150,900	368.700	80,000	480,000	100,000	100,00 & up	100,000
Typical cost, dollars/character	.0044	.0348	.0062	.007	.0068	.09	.0053

FIGURE 4

the heads. Complex mechanisms have been developed to handle the positioning, floating heads maintained by air cushions to protect the discs and positive measures provided to ensure separation in the event of power failures. The problems, however, are not completely solved, and movable head discs remain chiefly as large capacity storage devices with the disadvantage of excessively large access times. In the early discs devices the arms moved in unison, while newer units permit individual movement. A typical late model is the Data Products Discs being installed in the Q-32 Time-Sharing System, in which the access arms are mechanically independent, although, at present, only one arm may be positioned at a time. Careful allocation may decrease the effective access time, although the apparent mean will not decrease. In the Data Products unit, the access arm consists of eight heads; four above and four below which read or write in a simultaneous serial-parallel mode. The rotational delay encountered in the drum is also present in the disc storage unit. The movable head disc does provide an excellent random access storage for large data bases that would saturate the practical drum system.

A recent development in mass storage devices is the fixed head disc which may be thought of as a three dimensional drum storage surface. Access times are comparable to fixed head drums, in the order of 20 milliseconds. Due to the large number of characters handled through the basic control unit, the cost per unit character is comparative with high speed drums of a lower capacity. In the commercial unit,

the Burroughs B472, the transfer rate is quite slow, 100K characters per second. This results in transfer times almost an order of magnitude higher than those commonly available in drums. Undoubtedly, serial-parallel transfers would increase the transfer rate, but this would also increase the cost to the point where it equalled at least that of high speed drums. The fixed head disc should be thought of as the logical successor to the high speed fixed head drum, rather than a major breakthrough.

4.4.3 Magnetic tapes

The lowest cost device with the greatest capacity is still magnetic tape. The chief disadvantage, which proves to be decisive in most cases, is the lack of random access. Records are available only sequentially, and intolerable delays will result if an appreciable tape searching is required. The high density tape is capable of speed in the vicinity of 62.5K characters/sec. If the tape does not require positioning, access times are in the order of five to thirty milliseconds, which is comparable to other types. However, unless only two programs are active repeatedly, searches are required and access time in the range of tens of seconds encountered. This would obviously degrade a system to the point of almost uselessness. Tape does still provide an excellent storage for programs that are not active but need to be retained in the system. An example of this is the handling of disc overflows, where programs selected by some aging criteria are transferred to tape when active users require more disc capacity than is available. Tape,

therefore, appears useful primarily as a backup, or secondary, store and should be considered as a primary store only if the other types of more effective storage are not available, and/or severely reduced system performance is acceptable in light of the cost reductions.

4.4.4 Cartridge units

A promising storage device is the cartridge unit with replaceable sections which present a combination of the random access disc and the large volume characteristics of tape. Large delays are encountered when non-loaded cartridges are required, but this appears to be a method of handling disc overflows that should not be overlooked. Rather than transfer selected programs to tape as space is required, cartridges would be switched and retained for future use. There are problems in implementation, but the approach deserves consideration and careful observation to allow inclusion into the multiprogramming system when performance warrants it. If positioning times can be reduced, this device will be able to compete with discs in the large volume storage area.

4.4.5 Woven screen

The woven screen memory concept now under development is the multiprogramming storage device of the future and would solve many of the program exchange problems. A large capacity high speed random access storage device would be available at a relatively reasonable cost estimated at only about three or four times that of a high speed drum. It appears that the reduction in swap time, if any is required at all, will provide a sufficient time saving to justify the use of such

a storage device in terms of the great increase in the number of active users permitted.

4.4.6 Comparison of storage devices

The characteristics of representative units of the various storage device types are shown in Figure 4. The woven screen memory is included only as a forecast and will not enter into future discussions. It should be noted that the transfer rates of discs are only two to four times that of drums, while the access times are at least one order of magnitude greater. If schemes can be found to reduce the effective access time, discs could compete favorably with drums. The ability to issue a "look ahead" command followed by reading provides one scheme and will be fully explored. The higher cost per character of a drum requires that the size required be carefully determined. Fig. 3 provides a graphic presentation of the primary areas to which each storage device applies in terms of cost, typical sizes, and access time. These several descriptions provide an idea of the advantages both practical and economic which accrue from the proper combination of external random access storage devices. The basic characteristics described will be applied in any determination of the optimum exchange methods.

4.5 Memory Protection

Any practical multiprogramming system should provide a memory protection capability. During active periods, a minimum of two users, the Executive and one object program, must coexist in memory. If the

system is to function in a reliable fashion, the status of the Executive must be preserved unaltered. Further, to provide satisfactory service to the users, they should be protected from each other.

Two basic degrees of protection are readily apparent. The first, malfunction protection, is absolute in nature, encompassing both hardware failures and interprogram interference and appears virtually impossible. A comprehensive automatic recovery program such as the IBM FIX used in the Q -32 can overcome many hardware failures but requires extensive overhead and adds immensely to system complexity. This section will be concerned only with the second type of protection, interprogram protection, and the methods of attaining protection in an effective but flexible mannner.

Before deciding on a method, the degree of protection must be established. The Executive Routine must have access to all areas, whereas object programs should be either read or write protected or both, and illegal entries (jumps) should be prevented. Although it does not concern the Exchange problem, the I/0 Dispatcher must prohibit write operations in forbidden (Executive) or inassigned areas of both external storage and core.

The memory protection method and its effectiveness are very important parts in determining the overall system performance. An uncontrolled program can not only damage itself, but other users and the Executive routine which renders the system inoperative. In as much as the memory protection provided affects the program exchange

method used, it also exerts an influence on the number of users permitted. This section will describe several memory protection schemes in detail. Later sections will demonstrate the influence of memory protection on various exchange methods.

Due to the high frequency of core references in any program, protection should be implemented by hardware rather than software. The following list of characteristics is a modification of a list proposed by E.A. Codd (4) and represents the primary areas with which a memory protection scheme should concern itself.

1. Resolution - The smallest block which can be protected. Flexibility in determining block size should also be considered.

2. Adjacency - Can non-adjacent areas be simultaneously protected? What are the limits on this multiple protection feature?

3. Types of protection - Are data handling and operational violations handled differently? What combination of read and/or write protection is afforded?

4. Performance - What penalty, if any, is paid for protection in total system performance?

5. Treatment of potential violations - How are violations handled?

To provide a useful service to the user as well as a safeguard to the system, potential violations should be trapped and an interrupt jump to an error routine initiated. The user should be informed, as specifically as is compatable with allowable overhead constraints, of the nature of the violation. This general treatment of handling violations will suffice

for this section, and attention will be devoted to studying the ways various memory protection schemes are implemented and how effectively they perform.

The Bounds Register approach is one of the most useful protection schemes. It is controlled by the Executive which sets limit registers to bound the area available to an object program. Hardware comparison is made to these registers automatically and simultaneously for each instruction requiring memory access. This technique is used on both the IBM Stretch and 7090, the CDC 6600, and the RCA 601. The RCA 601 also utilizes the lower limit register in conjunction with object program addresses to provide relocation on loading. This method is extremely flexible in both size and location, and multiple bounds registers may provide protection to several non-adjacent areas. Another addition to the hardware permits selection of protection of the area either inside or outside of the bounded area. The disadvantages are the large amount of logic circuitry required and the timing problems involved in comparison.

A modified bounds register is used in the IBM 7040. In this approach, the lower boundary of protection is loaded into a 9 bit register, which is then compared against the higher order bits of the effective address to determine if the number is equal or unequal. Another register defines how many of the higher order bits will be compared. The value in this count register determines whether the protected area enclosed, starting from the base address, is 1 K, 2 K, up to 64 K. An additional

feature is the option of specifying the legality of the unequal or equal condition. One choice, the illegal inequality, protects the area outside the defined area. Conversely, the equality being illegal protects the area inside the boundary. Less hardware is required by this method than in the complete boundary register method, and timing considerations are not as stringent. Due to the bit setting nature of the approach, the size of the protected areas is is 2^n K increments.

The mask register method of protection is similar to limit register approach. The mask register has one bit for each memory block, and the setting of any bit establishes a protected area. Fewer hardware complexities and timing problems areise, as the method requires only one bit per block. The Executive can maintain a table of memory areas, or they can be carried with the object programs, and each change it controls will be preceded by a resetting of the protection mask. The system provides for any combination of blocks desired, but the size of the blocks is an inflexible hardware parameter.

The Ferranti Leo III uses a mask type of protection but applies it to each individual storage location in the form of a tag. When a program is entered into the memory, a tag identification is set on all locations available to the program. Object programs may use only those locations tagged for its use. Certain areas available to several programs are given special tags. This method is effective in the environment where several programs are placed in core and remain there until completion. Non-adjacent areas are protected, and any size protected area is

[illegible]

[illegible]

[illegible]

[illegible]

selectable. However, the method becomes consuming for applications where frequent exchanges are encountered and is generally unsuitable for time-sharing applications.

The Q-32 uses a mask type of protection scheme. Each of the four logically separate 16K memory banks has a control flip-flop to establish a protected area. The primary disadvantage is the coarse, 16K resolution. This will be improved by a modification to provide mask registers for each block. Protection will be available in contiguous increments of 2K with no non-adjacency provisions.

The next major approach is hardware lockout. In the Atlas application, object programs operate in a different mode than the Executive and cannot generate addresses addressing a restricted portion of memory. If, due to hardware errors, illegal addresses occur, interrupt transfers to the error routine are made.

Fixed or "read only" memories provide a rigid protection which requires little overhead or parallel logic. Deposited capacitors or inductive arrays are pre-set and cannot easily be altered. Compactness and low power requirements enable high switching speeds to be attained. This provides an excellent storage for the Executive but cannot even be considered for interprogram protection. Due to the difficulty in changing the storage, it is of doubtful value for even the Executive.

The last program protection method is a software approach. This is used in some debugging routines when no other methods are available, but it is so time consuming as to preclude its use in most other areas.

The most practical protection scheme from the Program Exchange viewpoint is a modified mask approach. The requirements for protection of blocks of under 1K in size cannot be justified in the practical case. The hardware costs, overhead impose, and circuit complexity far outweigh any increase in capability. The masks would be re-set each time control is shifted. The containment of the program in control provides the desired protection, and no further non-adjacency requirement is apparent. This is by no means a complete solution, but adequately defines a practical method which satisfies the general requirements of memory protection for the purposes of Program Exchange methods.

4.6 Relocation

The term relocatability can be defined as the independence of the object program from the constraint of occupying a specific area in memory. Relocatability provides a technique for improving the efficiency of program exchange schemes over the basic run and reload method. Core space allocation, which helps provide the desired improvement, is realizable only with relocatability. In most multiprogramming systems neither the system nor the program will know what area the object program will occupy at run time nor should they be required to. Indeed, due to exchanges, a program may occupy several areas during its execution. Combinations of hardware and software are best suited to providing the desired relocatability.

The most common method of program relocation uses a base register or pseudo base register. At first inspection, compilation to a base

address of zero with modification in accordance with a base register at load time seems to solve the problem simply and effectively. In principle, it does, but, upon closer investigation, problem areas are revealed. The various instructions treat the address portion in a great variety of ways, and no general modification scheme is apparent. Hence, each instruction must be inspected and then modified in the proper way using the base register. If a "relocation" bit, which signals the addition of the base register, is provided, much of the hardware complexity is avoided. This bit can be either part of the instruction itself or entered in the header of the binary record used to control the load.

Another approach is a basic system scheme which, incidently, provides relocation. This is page turning, developed to a great extent in the Atlas system. Pages of a fixed size (512 words in the Atlas) are transferred, and a page address register containing the most significant bits of the core address of the page loaded is associated with each page read into memory. Hardware comparisons of the required page address and the available page addresses are made to determine if the referenced page is located in core. If not loaded, an interrupt is generated and the page loaded as soon as possible. The hardware utilized for paging also provides memory protection. The program relocation problem is automatically handled through the page address register, and memory allocation is simplified. The paging concept will be treated in greater detail when studied as a basic program exchange technique.

While no particular scheme is favored, it will be assumed that relocatability is provided by a specific exchange method when required.

5. Complete Program Exchanges

As a basic exchange method, the complete program approach has the chief advantage of simplicity. Programs are handled in a natural fashion, in their entirety, and no undue overhead is created. The entire program is available at all times during the quantum, and closed subroutines can be utilized to reduce program size. No unusual demands are placed on the compiler as, at worst, only relocatibility bits are required. In contrast to the "page" approach, there are no inefficient pauses while pages are checked and required pages called, nor is a large overhead required to keep track of the pages and their status. Space allocation, memory protection and relocatability provide the most efficient type of operation. The most serious disadvantage is the presence in valuable high speed core of large strings of coding which, although they cannot possibly be operated on in one quantum, must be transferred in and out of core during each active period. Several large users can effectively thwart space allocation and make the value of memory protection and relocatability marginal. A relatively small core and a large Executive Routine which must remain in core at all times impose a limit on maximum program size. This can become a major system constraint in an environment characterized by large programs.

5.1 Storage Compacting

Prior to handling specific methods, two fundamental problems inherent in all program exchanges will be discussed. These are drum compacting and preservation procedures. A running measure of the

[illegible]

[illegible]

[illegible]

external storage must be maintained to handle new arrivals. However, due to a previous user becoming inactive, the available storage may well consist of broken blocks. The basic question is then, when to compact or compress the storage. Attempts could be made to insert a new user in available areas or immediately upon ascertaining that sufficient total space exists, the storage could be compacted and the new user placed at the end of the active programs. Inasmuch as the system envisioned would be open ended as far as users are concerned, it appears little would be gained by attempting insertion. The inspection of all open areas to determine if the size is sufficient would be time consuming, and subsequent arrivals would most probably necessitate compacting in any case. The general compacting philosophy will call for compacting whenever it is determined that a new user can be accommodated into the system, and insufficient room exists after the last active program on the external storage device.

5.2 Preservation of Environment

The second problem concerns the preservation and storage of the program environment, the information concerning the state of the object programs such as operational registers, I/0 control words, etc.. This information must be preserved for each program exchanged and prior to restarting all information restored. The actual operations are to be handled by the Executive, but the storage location is not as simply handled. If storage is to be maintained in the Executive, the table capacity must provide for the maximum number of users. If the system is large,

valuable core space will be lost to preservation tables using this method. The alternative is adding storage to each program which carries the required information. This increase can be implemented most effectively by the system at load time, although the compiler could add the required space. The first method permits the Executive to be re-setting the operational registers while the program is being reloaded, but appears to be very wasteful of core, especially in large systems. The time required to load several registers will, however, be relatively small (20 or fewer major cycles). In view of this and the more effective use of core permitted, this method of status preservation is recommended. The general concept has been used in both the SDC TSS-2 and CDC 6600. A proposed preservation routine for the CDC 1604-160 Satellite System is presented in Appendix III.

5.3 Basic Complete Program Exchange

The fundamental complete exchange method is the run and then dump and reload procedure. Its lack of sophistication provides its greatest virtue, simplicity. No space allocation is implied, and, with only the Executive requiring protection and no relocatability required, the hardware problems are simplified. Users are loaded on the external storage, typically a drum, as they arrive, and the capacity of the drum determines the maximum physically allowable number of users. As only one object program is permitted in core, the maximum number of users for a given reaction time can be obtained from

$$n_{max} = t_r(1-\eta)/\ 2t_s+q$$

which is the worst case form previously developed. Lack of concurrency of t_s and q seriously degrades the system performance. In addition, during the period $2t_s$ for each user, the central processor is idle, seriously reducing the computing efficiency. In a system where the number of active users per cycle, a function of both the number of on-line station, N_{max}, and the job mix, is low, this type of exchange is valid. If the smaller n_{max} is acceptable, the simplicity of the approach recommends it. This approach will serve as the basis for other complete program exchange methods and attention will be devoted to various means of improving the efficiency and overcoming the disadvantages.

5.4 Two Level Storage

The use of drum or fixed head discs seems natural in the basic case due to the transfer times, and an increase in system capability can be achieved by addition of a movable head disc with its large volume storage, albeit slower access times. In the most common usage, the disc replaces tape storage and speeds system operation but has no effect on the exchange problem. The disc, though, can serve as an effective secondlevel storage device. Although the actual scheduling will not be covered, the disc could handle large programs such as compilers, whose long run time and few man-machine delays permit a reduction in response time. To prevent the introduction of several compile type jobs from degrading the system due to several repeated slow disc accesses, a reduced service interval could be used for disc programs. If only one disc program was allowed per cycle with

all programs on the disc receiving service from a round robin queue, system response would suffer only slightly and virtually no delay would be noticed. Using this approach, the compute limited jobs would receive service while not overloading the system drums This type of treatment could also be applied to slow background or batch type jobs. A limit should be placed on even this service, as the primary function of the disc is to overcome tape deficiencies. The secondary object program storage should not be permitted to curtail this to any great extent. The overhead involved can be justified in view of the increased service offered by the system and the increased storage made available on the drum for generaly smaller, fast reaction time programs.

5.5 The Disc "Look Ahead"

In a drum and movable head disc system, the largest delay encountered is in positioning the disc heads. Reduction of effective access time increases the efficiency of the concept. Means do exist through "look ahead" features to reduce this time. Discs are available which permit the issuing of position commands only, and the transfer operation is ordered only after the heads are positioned. Considerable time could be saved if the positioning could occur concurrently with the running of another program. Then, when the present q was completed, the next program would be ready for loading. In current units this approach has the serious disadvantage of precluding the simultaneous use of the disc, which was basically to replace tapes, by an object program. The increase in system capability offered by use of the "look

ahead" must be carefully weighed against the lack of disc reference for short periods. The true effect on object programs involves the theory of program structure which is beyond the scope of this paper, but the handling by use of a WAIT command is a method. The WAIT reply, however, can cause one or more programs to loose their entire useful quantum, and, due to their remaining active until serviced, probability considerations show that the situation is likely to deteriorate with no one receiving good service. For this reason, the method is not recommended for general usage. The alternative is the hardware capability of independently positionable movable heads or fixed heads. Either of these add considerable cost to the system but provide a powerful capability. If the economic considerations warrant it, one of these features should be included.

5.6 Complete Program Exchange with Space Allocation

The three previous techniques, while providing increasingly better service, are still constrained by the

$$n_{max} = t_r(1-\eta)/ \ 2t_s+q$$

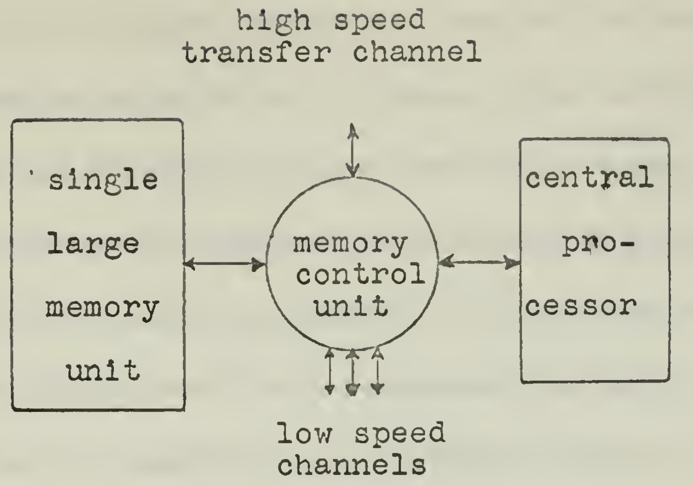
equation. Space allocation or space sharing of core provides the most practical method of improving this user factor. The concurrency of exchange and run operations permits the reduction of effective swap time, and permits either more users or better service to the same number of users. As previously developed, this method requires the hardware for relocation and memory protection.

5.6.1 Hardware constraints

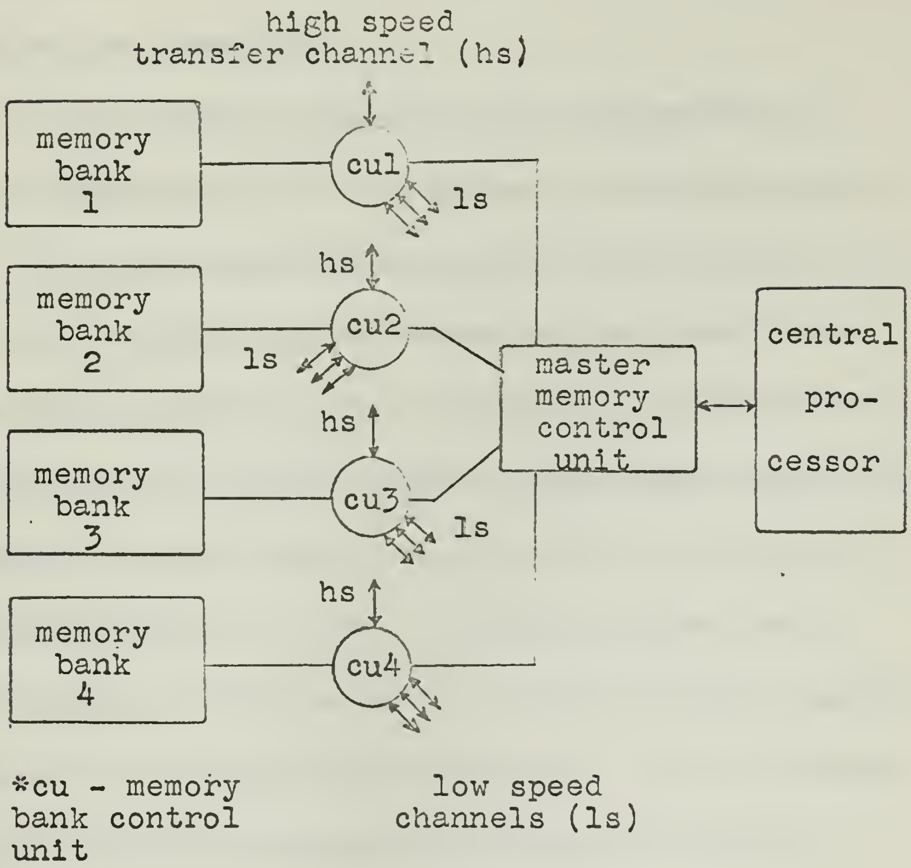
Concurrency of exchange and run modes will focus attention on the high speed channels, and some introductory material is in order. The subject of high speed transfer channels presents some interesting paradoxes. High speed is generally construed to mean at a speed comparable to one memory cycle. Due to the read/write nature of memory references, it can be seen that two high speed channels cannot operate simultaneously in a given core unit. The slowing down of the channels and inter-leaving of operations does not improve speed, but only ensures that each job will require $2t_s$, and that they will be completed at about the same time. An alternative is the use of multiple logically independent memory banks. A separate high speed line can be provided for each bank and a simple stepping mechanism used in the memory control unit to allow large programs to utilize several banks. The method also facilitates memory protection, especially at the block level. Fig. 5a and 5b depict the basic configurations. While the second has a higher cost due to both the more complex control unit and the multiple high speed lines, its advantages are attractive.

5.6.2 Hardware determined methods

The extent to which space allocation is implemented depends upon the number of simultaneous transfer operations permitted, a hardware determined parameter. If the three operations - run, load and save - procede concurrently, a smaller effective exchange time results if the mean program size is one third the available memory or less. A quantum



5-a
SINGLE CHANNEL CONFIGURATION



b
MULTI-CHANNEL CONFIGURATION

HIGH SPEED TRANSFER CHANNEL CONFIGURATIONS

FIGURE 5

at least as long as the time to transfer one third of memory is presupposed. If only one transfer operation is provided the acceptable mean program size increases to one half the available memory, and the minimum quantum becomes the time to transfer the entire core. The quantum constraints are introduced to reconcile the general timing problem. If very short quanta are allowed, it is impossible to complete the exchange concurrently with the run operation, and little improvement, if any, can be realized regardless of the exchange method. The quantum must be as long as the time required to perform the complete exchange of programs of mean size.

5.6.3 Space-sharing exchanges

The use of space-sharing in the full program concept does not assure increased effectiveness since the allocation algorithm increases the overhead. If the mean system program size is a large fraction of the total available core, transfers will be required at almost every quanta. The method will then proceed similarly to the basic run and reload method, but will require the extra overhead to check for space-sharing possibilities. A similar condition can result if many users are present in the system and the size of the available core is small. If the mean program size is greater than the average allowed per user, the system becomes saturated and little is gained from attempted space-allocation. The latter problem can be overcome if concurrent transfers are permitted but should be carefully considered if this concurrent capability is not provided.

The use of space allocation in conjunction with both the disc and drum storage provides the maximum complete program exchange efficiency. This maximum efficiency requires several conditions be satisified. First, the mean program size for drum users must be less than half the available core memory. Second, hardware must exist to provide at least one level of concurrency of exchange and run modes. The previously mentioned requirement of relocatability must be met, and memory protection of some sort is advisable. The disc would be used to store a limited number of large programs, scheduled in a slower manner than the drum users. Failure to satisy any of these requirements will detract from the method, and a thorough study should then be made to see if a less complex method would not provide as efficient overall service.

6. The Block Exchange

The block exchange technique provides several extremely useful features to the time-sharing system. Block exchanges permit many users to reside in core concurrently and dependent on the program mix, several of the transfers required by the complete program approach are avoided. This reduces the average exchange time, t_s, and as shown previously, increases the upper limit on the number of active users permitted. If additional blocks are required, the exchange time is, at worst, no greater than that required for the entire program to be transferred. The penalty paid is in increased overhead which must be carefully controlled. Use of the proper algorithm minimizes "page" turnings, or the number of times a block is dumped and then reloaded. A further advantage of the block exchange method is that the programmer is virtually freed from the constraint of particular machine memory size. Programs written on a general Symbolic Language can be compiled with any applicable compiler without regard to memory size. Due to the relatively small size of the blocks, space allocation is more easily implemented than in the complete program exchange methods.

The block concept greatly increases the complexity of the compiler, as the blocks must be assembled and references to other blocks treated in a distinctive manner. Some systems use hardware design to simplify these problems, and those without the hardware capability must rely on the time consuming software methods. One of the most obvious ways to simplify the compiler is the liberal use of open subroutines which

[illegible]

reduce the number of program jumps. This greatly increases program size but, as this is no problem in the block method, it can be accepted. If references are made to blocks not in core, pauses, which degrade the particular program and decrease system reaction time result. In most applications, complex checking routines are also required to first determine whether or not a reference is in the same block, and, secondly, if it is in another block, to determine if the required block is in core and where. If this is not the case, the location of the block on the external store must be determined, and the block loaded. This latter, three phase problem, basically the excessive time and space required for maintenance and checking, is one of the most serious disadvantages of the block exchange. A second major disadvantage is the difficulty of access to large data bases. Successive references to widely scattered data entries generate numerous block calls which result in losses in efficiency.

6.1 The Page Exchange Method

One of the earliest uses of the block exchange method was in the Ferranti Atlas system [17,18]. The implementation in this case is extremely hardware oriented, and the discussion of the concept has value not solely as a particular method but further as an example of the capabilities that can be achieved by hardware design. It is indicative of the features that will be further developed in the next generation of large scale systems, and less sophisticated systems will attempt to obtain the same advantages through software. The

[illegible]

[illegible]

[illegible]

[illegible]

system solution to the space allocation problem will also be discussed. Although a complete explanation of the entire system will not be attempted, sufficient detail will be included to provide a background for the exchange method.

The storage hierarchy of the Atlas is rather unusual, and the exchange method derives much of its power from this storage approach. The primary storage consists of normal ferrite core, a high speed drum (2 msec/block) and an unusual storage termed the fixed store. The fixed store contains several thousand words of fast (300 ns) storage and consists of a woven wire mesh with small ferrite plugs inserted into the spaces. The presence or absence of a plug determines the state of the store. The fixed store holds complicated functions which permit both a simplification of the arithmetic unit and inclusion into the instruction code of complex instructions such as vector operations and polynomial evaluation. The core is divided into stacks of 4096 words, each with an independent access to the central processing unit. The stacks are interleaved in pairs, odd and even, and instructions drawn from the store in pairs. Transfers between drum and core are direct, in 512 word blocks, and, once initiated, proceed autonomously.

From an exchange viewpoint, the most valuable feature in the Atlas structure is the one level store concept. The core and drum are considered as a single large memory (maximum size 10^6 words) and can be addressed as such. In the following discussion some difference between Atlas terminology and normal usage requires clarification.

The Atlas system defines a block as 512 words of information and a 512 word memory unit as a "page" in core and a sector on the drum. Further, the term address refers to the identifier of a required piece of information and does not necessarily provide its location. The 20 bit address (Fig. 6) consists of 11 bits providing the block address and 8 bits providing the location of the word within the block. Each page of core has associated with it a 12 bit page address register, 11 bits of which identify the block contained. When a storage reference is made, a parallel comparison of the page address registers is made, and non-equivalence indicates the block is not in core memory. A suitable interrupt is then generated, and the Executive, termed the Supervisor in the Atlas system, initiates the required transfer. This frequent, complex checking requires involved system tables which are maintained in a subsidiary store accessible only by the Executive. The block directory contains an entry for each block in the one level store consisting of the block number, n, and the location of the page or sector occupied by that block. Each object program is assigned a distinct area, and a separate program directory defines the areas occupied by each program. The number of blocks required is one of the job input parameters.

During some operations such as drum or tape transfers, a page cannot be moved or accessed by an object program. To protect such pages a lockout bit is provided in the page address register. Reference to a protected page causes an interrupt to be generated, and the Executive takes the proper action. The capability further provides the Executive

ATLAS BLOCK

ADDRESS FORMAT

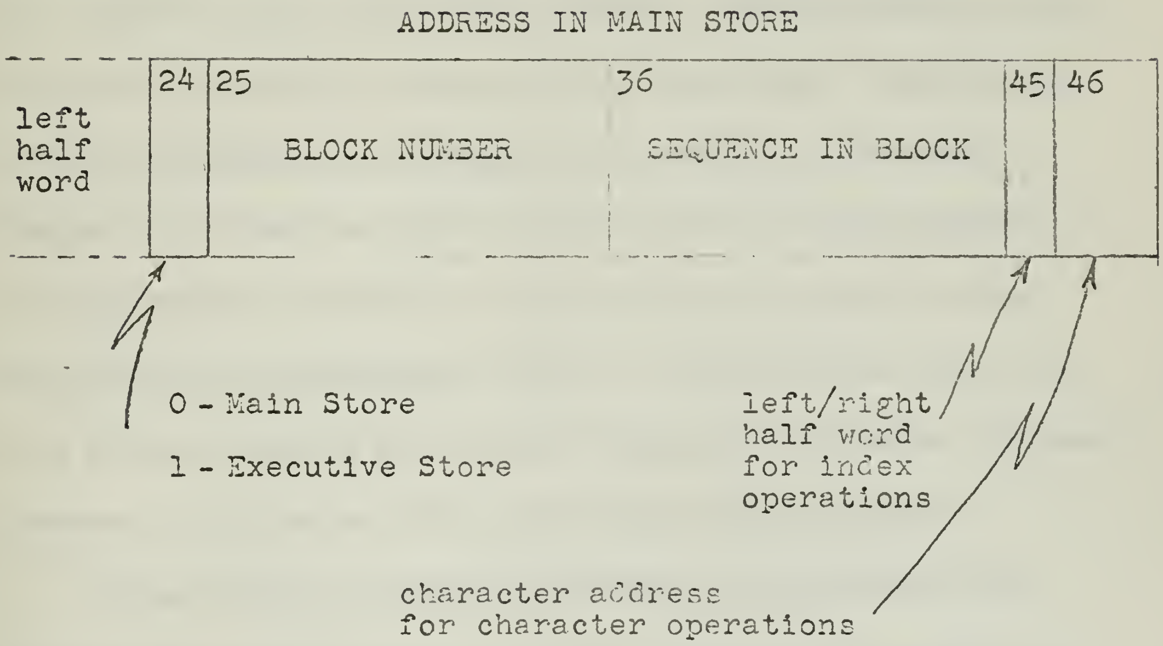


FIGURE 6

with a flexible memory protection technique and allows protection to be shifted easily as the controlling program is changed.

The Executive insures that the core always contains an empty page. When a non-equivalence interrupt is received, the Executive locates the required sector using the block directory and initiates a transfer to the empty page location. While this transfer is proceeding, another page is selected for transfer to the drum to fulfill the empty page requirement. The selection of the page to be transferred is made by an adaptive type program which predicts, learning through its errors, the page that will not be required for the longest time. Various types of adaptive programs to accomplish this selection are described in Section 6.2. When the initial transfer to core has been completed, an interrupt transfers control to the Executive which updates the block directory and program address register. The transfer from core to provide the empty page is then initiated, and, upon completion, the block directory is updated and the location of the empty page noted.

These storage and exchange techniques combined with other interesting hardware features provide the Atlas with an excellent multiprogramming capability. Representative drum times show access and transfer times of 6 and 2 milliseconds respectively. Due to the one level store and the reference by block address and then by location within the block, relocatability is not required as it is provided implicitly by the storage method. The program directory reduces the number of entries in the block directory that need to be checked for each

[illegible]

non-equivalence interrupt to determine the block location and, hence, effectively reduces overhead. The one level store and allocation scheme provides a definite advantage to the programmer in that program can be written without regard to the machine on which they will be run, especially as far as size is concerned. An interesting effect of the adaptive page selection routine applies to large multipurpose programs. If, for a specific application, only a few blocks are needed, they will exist in core with a high probability, and the unused portions will remain on the drum and not require exchange time. The system efficiency would be improved if the transfers to and from core could proceed concurrently. But the interleaving of the logically separate memory units, rather than the sequential treatment, precludes this, even if the additional high speed channel were available.

6.2 Pageturning Algorithms

The use of adaptive learning programs to select pages for replacement in core is based on the fundamental assumption that a good correlation exists between the previous activity of a particular page and its future usage. The determination is not a trivial problem and depends on a study of the program structure of the system in question. The arrival times and, hence, usage of core space of programs that require a short, one or two, quantum, burst of service followed by a long latent period awaiting human response are extremely difficult to predict. As service periods increase, however, prediction becomes feasible. Any so called page turning algorithm will increase overhead due to the

time required to record data and make computations. The probable effect must be carefully determined to ascertain that the overhead is justified.

6.2.1 Two level checking

In a short service environment, a two level page check would probably be satisfactory. First, a check could be made to determine if any pages in core are for users who are not active during the present cycle or, secondarily, have already received service during the cycle. If any such pages are encountered, as many as required could be exchanged. In the event that all pages belong to the remaining active users for the cycle, an arbitrary exchange of a random page would require less overhead and probably be as effective as any other determination.

6.2.2 Adaptive algorithms

As soon as any user or users require service for several successive quanta, an adaptive page selection algorithm becomes worthwhile. The algorithm attempts to minimize the turnings per unit time, or the number of times a page is required to be reloaded after being transferred out of memory. As pointed out previously, a poor choice, will, at worst, result in an exchange time equal to the complete program exchange time, t_s, plus the overhead, while better methods improve computer efficiency and system performance. The number of page turnings will be the basic parameter rather than page accesses which are more difficult to determine and less indicative of problems in system performance. An algorithm is required that not only considers the transfers of a given page, but has

some method of weighting and decaying so that a page heavily used during a previous period does not receive undue priority in the future.

6.2.3 The decay algorithm

J. W. Weil and J. W. Harriman have suggested methods of computing a figure of merit for each page which is adjusted when the page is transferred into core and updated for other transfers while it is in core. The overhead required to update all figures of merit for each page turning would be prohibitive, and a second parameter providing an indication of the last update would be helpful. With this parameter set when a page is turned out to the drum, no further action is required until the page is returned to core.

The basic algorithm proposed by J. W. Weil uses the following equation to compute figure of merit:

$$F'_{mi} = F_{mi}\, e^{-(t-t_i)\lambda} + \gamma\alpha$$

F'_{mi} = new Figure of Merit

t_i = last time F_{mi} was updated
t = present time
λ = decay factor
γ = 1 if page turned in
= 0 if page in core
α = new page emphasis factor

The time parameter could use the real time clock, but a greater sensitivity will result if the unit of t is a page turning, and t reduces in effect to a turning counter. The previous equations are reduced to:

$$F'_{mi} = F_{mi} e^{-\lambda} \qquad \text{for pages in core}$$

$$F'_{mi} = F_{mi}\, e^{-(n-ni)\lambda} + \alpha \qquad \text{for page just turned in}$$

The values of F_{mi} and n_i are stored either in a table in the page itself or in the status and environment storage for the program. When a turning is required, the page with the lowest F_{mi} is transferred and all other F_{mi} updated.

J. W. Harriman uses accesses to a "page" to update it rather than page turnings. This requires hardware sensing of an access different from a preceding access (regardless of whether the new page is in core). The individual accesses are used for the n and n_i of the previous equations. The variation requires a greater hardware capability, including the ability to differentiate between instruction and data accesses but results in an extremely fast adaptation and an efficient paging algorithm.

6.2.4 Probability allocation

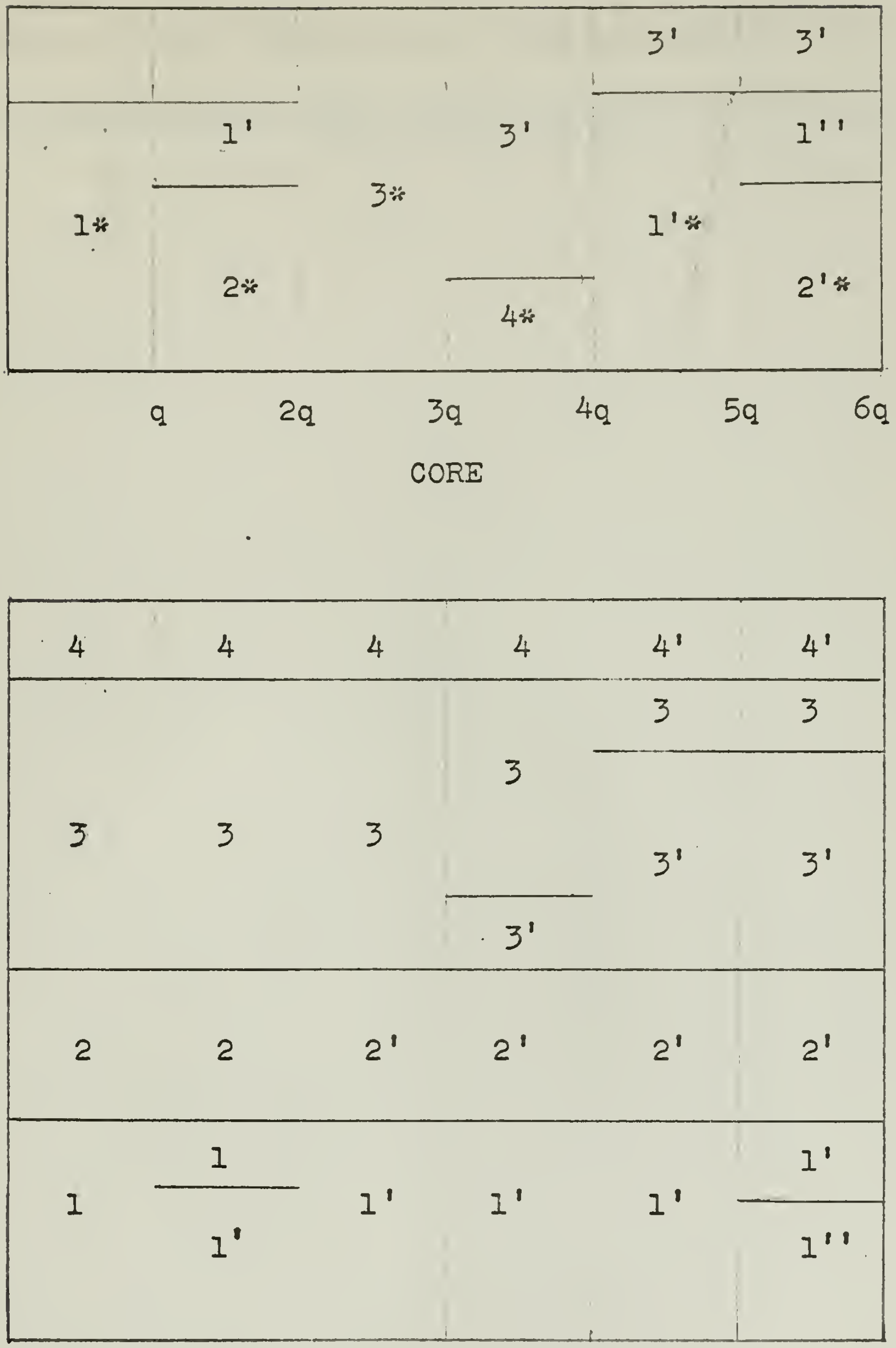
Probability allocation provides another method of increasing system efficiency by reducing the exchanges required. It has its greatest value when the system requirements can be defined to some extent. A typical example presented by B. N. Riskin deals with a data processing and multiprogramming system. The programs to be operated are known, and all possible combinations in a cycle can be tabulated along with associated probabilities. The process then evolves into one in which storage is allocated to a particular table or program in such a manner that other possible users occur with a lower probability. The fact that some programs or program parts must coexist for a cycle is also considered in space allocation. In a general time-sharing system where the program mix for a given cycle is extremely variable, the method does not seem practical.

6.3 The Pseudo Block Exchange

The pseudo block exchange method loads programs in their entirety, as in the complete exchange method, but uses a block type approach for transferring programs from core. When there is insufficient space in core for the next user, only that portion of core required by this next user is transferred to the external storage. If there are only a few active users of widely varying sizes, a considerable saving in exchange time can be realized. This method was developed originally for the MIT Time-Sharing System in an effort to overcome the problems of slow speed transfers in a core and disc only system. Although other systems utilize high speed drums to store active programs, the method is deserving of general consideration.

The overhead required to keep track of the location of various program blocks appears imposing at first, but reduces to a simple two entry table for each user. The first entry would provide the complete program length, and the second, a measure of the portion in core or on the external device. The program will exist as it did upon completion of its last quantum, partially in core and partially in the external store. Fig. 7 depicts the actual variation in the stores with the status depicted only during running quanta with exchange phases not depicted. It is conceivable that, at any given instant, parts of several programs, in addition to the complete program being run, would exist in core.

To simplify transfers, exchanges will usually be handled by blocks rather than individual words. The method is not a block exchange



EXTERNAL STORAGE

* indicates program operating during quantum, q.
' indicate successive states of a program(1 1' 1")

THE PSEUDO BLOCK EXCHANGE

FIGURE 7

in the usual sense, but provides a useful composite of the two general techniques. It could well serve as an interim exchange method until larger core and external stores could be integrated into the system.

7. The Completely Hardware Oriented Exchange

The final program exchange technique is quite different from both the basic block and the complete program approaches and was not even considered in the same general class. This technique is a completely hardware oriented system and the problems and potential of this approach are just now being realized.

7.1 The CDC 6600

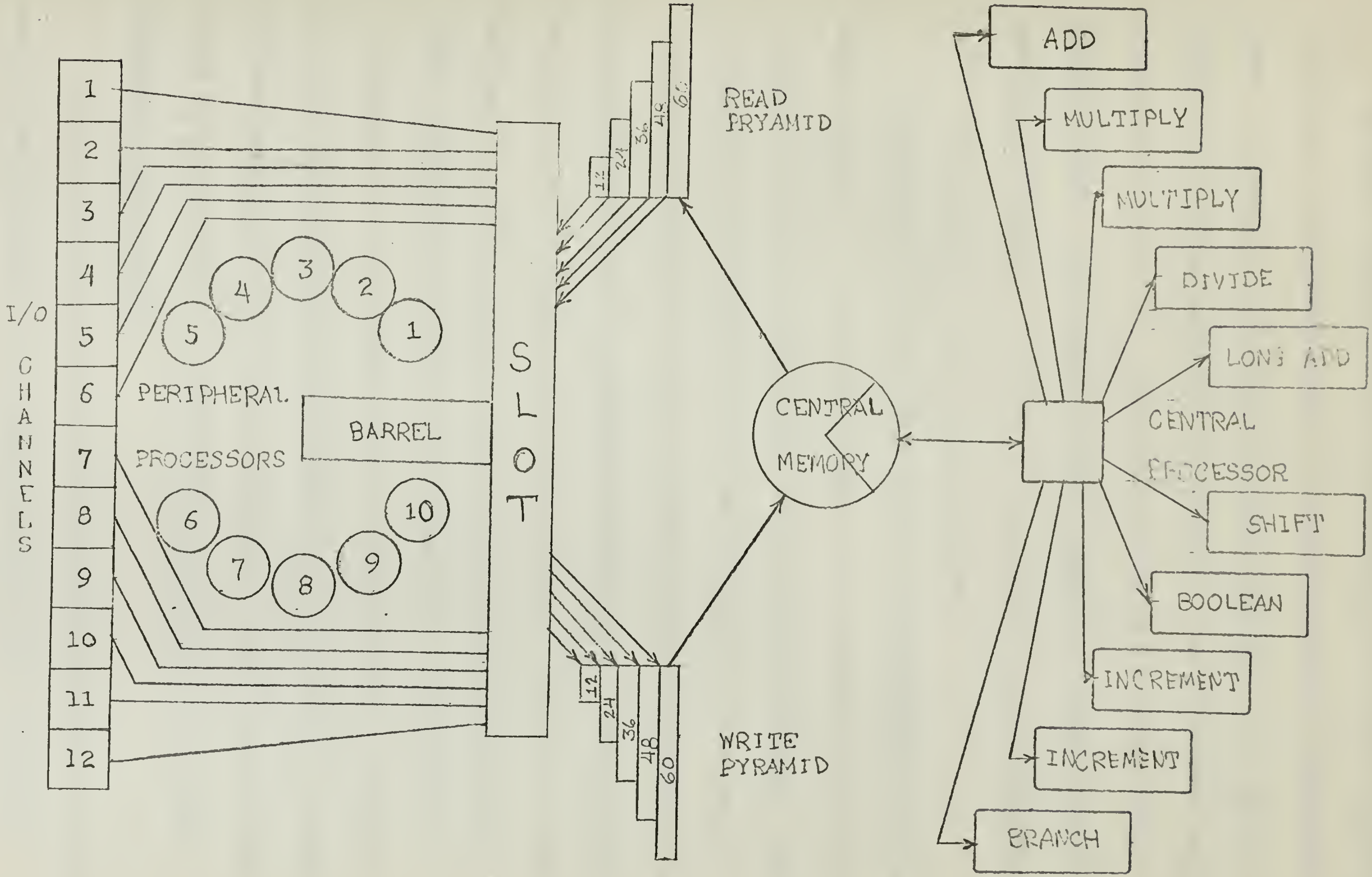
The recently developed CDC 6600 incorporates several of the most advanced techniques of both multiprocessing and multiprogramming. The methods utilized show how many of the complex multiprogramming problems may be efficiently handled by hardware design and point out trends in hardware system design. The multiprocessing capability is provided by ten peripheral and control processors and a central processor. Multiprogramming is handled both in the central processor and in the peripheral processor arithmetic unit, with the peripheral processors themselves acting in both cases as on-line users. The peripheral processors each have a small 4 K independent memory and are used to handle all I/0 transfers and to perform simple arithmetic operations. Programs requiring complex or high speed operations use the central processor on a time shared basis. The central processor operates on peripheral requests only and is not assigned to a specific program.

Before studying in detail the exchange method utilized, a brief description of the three major units would be of value. The characteristics of the central processor, the peripheral and control processors and the

central memory, in reality, determine the basic exchange method. Fig. 8 provides a basic block diagram of the 6600 system.

7.1.1 The central processor

The central processor relies on minimization of memory references to achieve a high speed. Programs to be run are stored in the central memory and initiated by an exchange jump instruction from a peripheral processor. The peripheral processors also establish upper and lower bounds to provide memory protection and specify the error exit to eliminate Executive overhead for illegal references. Multiple arithmetic and instruction registers are used to minimize storage references and increase the overall speed. In addition, multiple banks of memory permit concurrent referencing. The actual processor has ten different arithmetic units, each designed for a specific operation such as Boolean, multiply, etc.. Non-dependent instruction strings are sensed and may proceed concurrently, while a reservation system maintains the required program order. Programs are formulated in the normal manner, and the 32 instruction registers are updated frequently enough to maintain the program flow without pausing for memory accesses. Branch or jump instructions cancel the remaining instruction registers and they are reloaded. Central processor references to the central memory are made relative to the lower bound set by the peripheral processor exchange jump instruction and relocation is provided by simply modifying the lower boundary.



THE CDC 6600 SYSTEM

FIGURE 8

Hardware features cause the programming of the central processor to vary from conventional methods when examined at the machine language level. An example is the effect of multiple registers. The central processor has 8 operand registers, X O through X 7, and 8 address registers, A O through A 7; X 1 to X 5 read operands, while X 6 and X 7 write to central memory. The action is initiated simply by a change in the corresponding A register. A O and X O are used as scratch or buffer areas.

7.1.2 The peripheral and control processors

The peripheral and control processors are independent units each with a separate 4 K, 12 bit word memory. The instruction set includes access to the other two major units, I/0 and logical operations, fixed point addition and subtraction and indirect addressing. The ten processors use conventional programming techniques, but the instructions are executed on a single multiplexed arithmetic unit. The instructions are stored in a ten position barrel which can be thought of as a fixed form queue. As each position reaches the arithmetic control unit or "slot", all or part of the instruction is executed. If the system is considered in the time sharing context, each processor is basically an on line user receiving a fixed quantum (100 ns) of service. A concurrency technique permits memory references to be serviced and made ready for service while the instruction is moving through the barrel awaiting its next quantum.

All data transfers are conducted through the peripheral processors with an initial transfer to the peripheral memory and a subsequent

[illegible]

transfer to either the external devices or the central memory. Transfers to and from the central memory are conducted through read/write pyramids. The pyramids assemble and disassemble 60 bit words in 12 bit segments per stage. A position for 60, 48, 36, 24 and 12 bits is available in each pyramid so that four transfers can be proceeding in each direction simultaneously. The actual transfers from the pyramid to the applicable memory are handled through the "slot". Fig. 8 demonstrates the basic flow in the time sharing of the peripheral arithmetic unit. Communications between any peripheral units and/or external devices are handled on 12 bidirectional I/0 channels with a maximum transfer rate of 10^6 words (12 bits) per second.

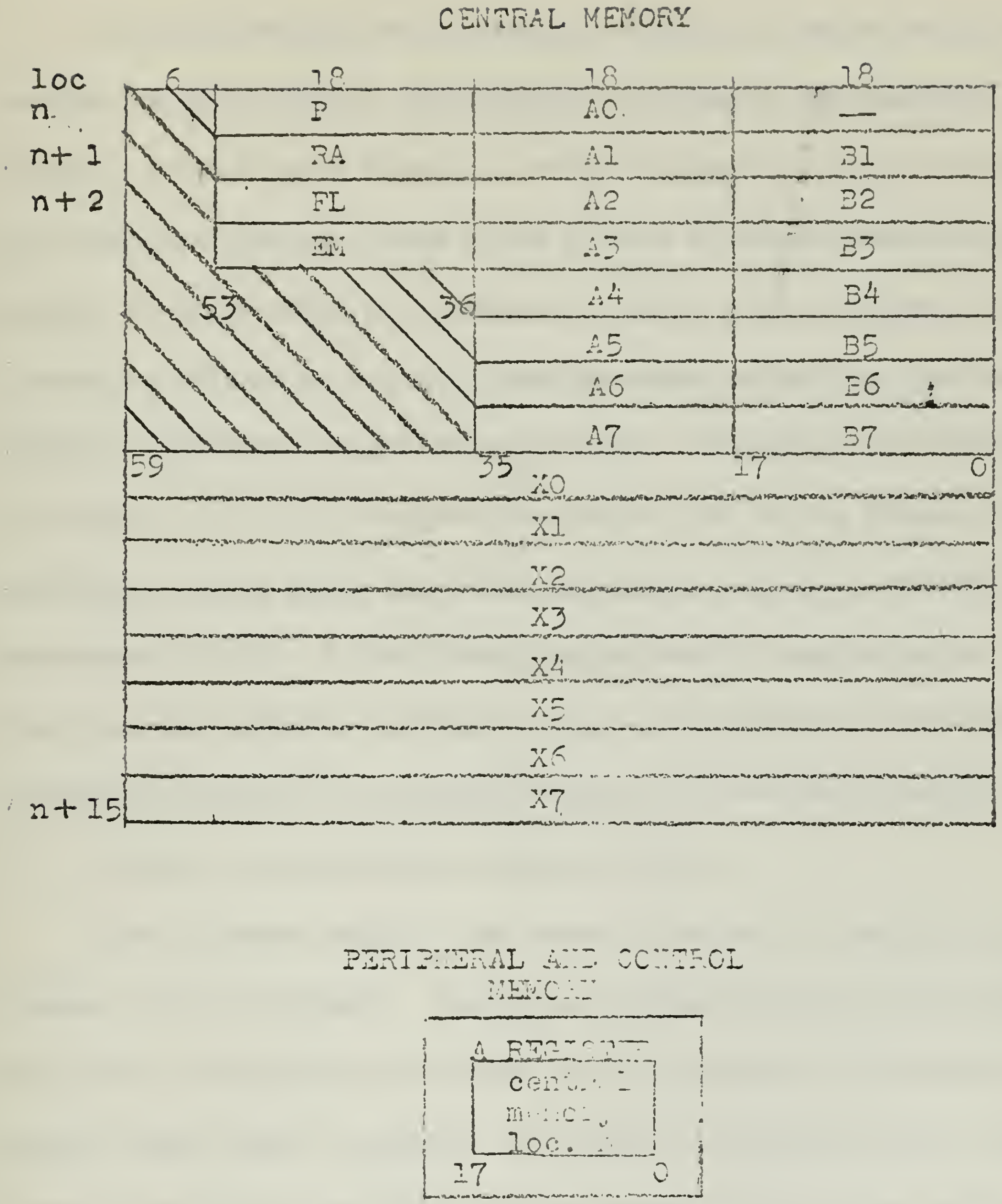
7.1.3 The central memory

The central memory consits of 131K words (60 bits) organized in 32 logically separate interleaved banks. Consecutive addresses go into different banks permitting the rapid loading of the 32 instruction registers in the central processor. The address and data control mechanism permits a transfer rate of 10×10^6 words per second. A novel control technique is used to handle all references. A clearing house called the "stunt box" receives all requests, and, under a priority system, passes the addresses on to all banks. The applicable bank accepts the request if not in use and notifies the "stunt box" which then initiates the transfer to the data distributor. If the bank referenced is in use, the request is stored in a hopper. Addresses are sent from the hopper, central processor and the peripheral processors in that

priority and repeated until accepted. Division of four banks use a common line to the data distributor which serves as the transfer point.

7.2 The Exchange Jump

An exchange jump instruction provided in the peripheral processor repretoire initiates program operation on the central processor. The instruction first generates an interrupt and transfers the program starting address in the central memory to the central processor via the accumulator of the peripheral processor. When the interrupt is sensed, the program status and environment are set in the central processor's registers and the information from the previous program saved. A subsequent exchange jump instruction referencing the block stored returns the interrupted program to the central processor for further service. The format of the exchange blocks and a description of the registers involved is given in Fig. 9. Once the status is set, the instructions are loaded into the 32 instruction registers and execution commenced. All central processor memory references are made relative to the reference address, one of the status registers, permitting easy relocation. The upper boundary of memory protection is established by the sum of the reference address and the field or program length. The program address register P is an index relative to the reference address, and the P = 0 word is used for program exit condition storage. An exit feature permits programmer selection of the exit and stop conditions of the central processor.



P = Program address   A = Address registers

RA= Reference address   B = Increment registers

FL= Field length   X = Operand registers

EM= Exit mode

THE EXCHANGE
JUMP INSTRUCTION

Figure 9

7.3 Executive Control

The 6600 provides an outstanding example of a hardware implemented Executive routine. The functions required by the Executive, as previously described in Section 3, are all handled to provide a virtual two level time-sharing system which utilizes an unusual exchange technique. No actual transfer of programs is made, and the exchange jump instruction only saves and loads program status and environment information and switches control among programs. Similarly, the multiplexed arithmetic unit for the peripheral processors uses the ten independent memories, and the barrel and "slot" provides a method of switching operational control. A lower bound register and a computed upper bound memory protection scheme is used with relocation for the central processor provided by the reference address or lower bound register.

7.4 Critique of the Hardware Oriented Method

The only major fault is to be found is the lack of flexibility and general system complexity. Due to the hardware nature of the system Executive, all procedures are fixed, and no capability of adaptation to special users needs is apparent. To obtain the maximum benefit of the powerful techniques available, the programmer's job is made considerably more difficult, although this could be remedied by a comprehensive compiler and system control, or "Monitor", program. Another weakness seems to be the relatively small memory. If the ten peripherals are presumed active, an average program length of only 13.2K is permitted. The effect of this constraint is, of course, dependent upon the job mix.

On the whole, the system does provide an impressive portent of future generation multiprogramming systems, and the methods used undoubtedly will influence greatly future time-sharing systems.

8. Simulation

The use of a system simulation program provides another technique for analyzing Exchange methods. Further, it permits investigation of the effect of varying certain basic parameters on various Exchange algorithms. A time-sharing system is a stochastic system and is amenable to the Monte Carlo method of analysis. The arrival of users and job characteristics can be treated in a probablistic sense and used to supply inputs to the system. The manner in which the various Exchange algorithms service these users can be used as a good approximation of actual performance in an operating system. Simulation provides a valuable secondary result in system planning. The coding required to simulate an Exchange algorithm approximates very closely the actual coding to be used in an operating system. Thus a measure of the complexity of the procedure is obtained, and, more importantly, the overhead attributed to the various methods can be closely approximated. Inasmuch as the overhead is a critical factor in the comparison of Exchange techniques, accurate determination of overhead time is invaluable.

The general development of the system simulator, Program SIM, is covered in detail in Appendices I and II. This chapter will treat the Exchange portion of SIM specifically and will present and analyze the results of several simulation runs using different Exchange algorithms. The changing of either the Exchange technique or any of the system parameters permits many diverse systems to be investigated. The

[illegible]

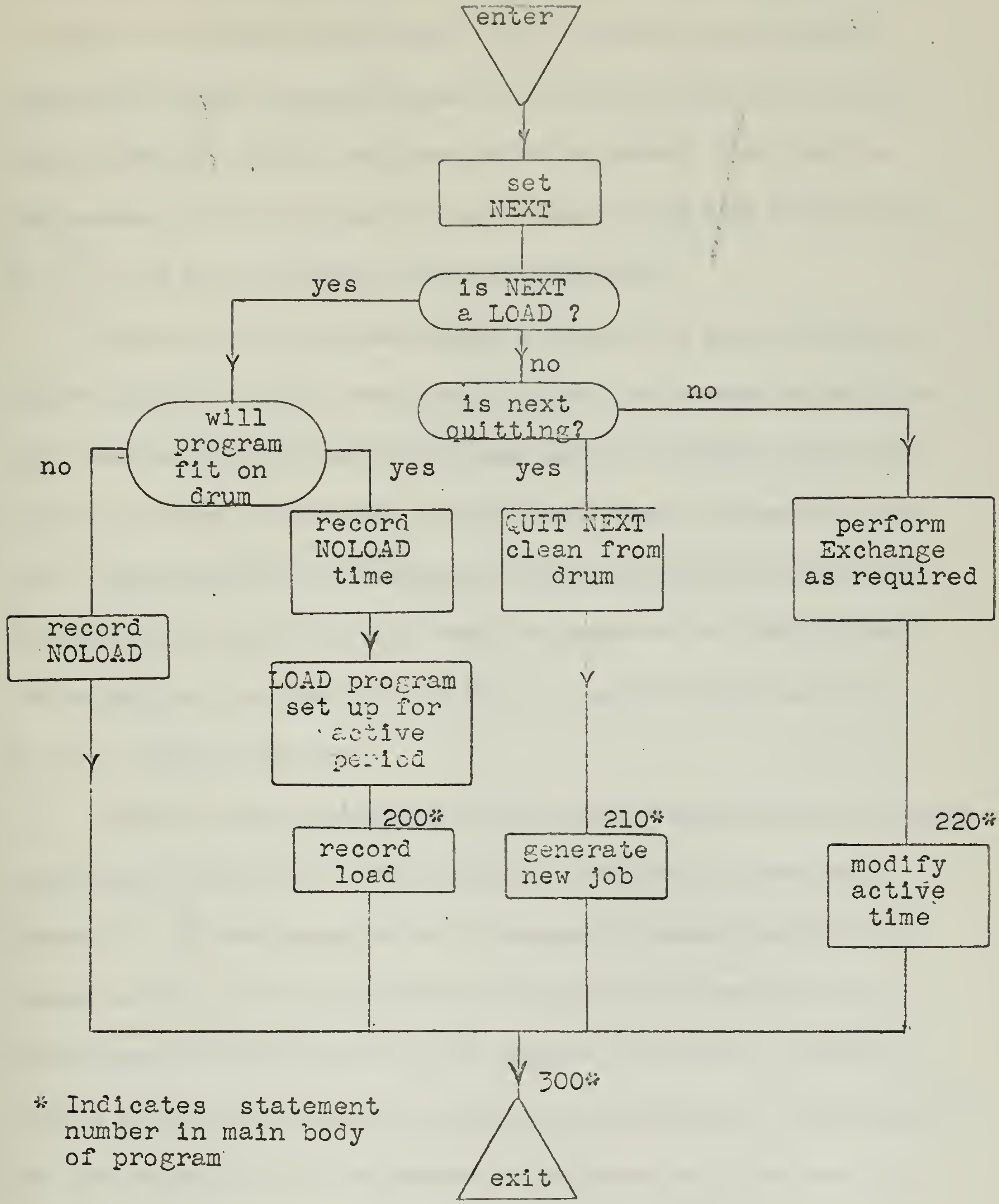
[illegible]

system parameters include core, drum and disc storage capacity, number of users allowed and mean program size as well as the other job parameters. A computer with a three microsecond cycle time is assumed and transfer times computed on this basis.

8.1 Simulator Exchange Routines

The Exchange portion of SIM is divided into three major parts corresponding to the three basic system functions handled by the Exchange routine. These are the LOAD, QUIT and EXCHANGE operations. LOAD and QUIT bring the binary program from permanent storage, such as tape, to the external storage device used to handle operating programs and remove a completed program from this external store, respectively. Bounds are placed on the capacity of the external store, and NOLOAD conditions may arise due to lack of space. Figure 10 provides a flow diagram of the LOAD and QUIT routines. To permit completed programs to be saved if desired, upon receipt of a QUIT command the program is transferred from core to the external store before actual termination of the users service. Regardless of the Exchange algorithm used, the LOAD and QUIT operations are the same. The rearrangement or compacting of the external store was discussed in Chapter 5 and is not considered in SIM. It is felt that this is somewhat of a Dispatcher problem and would only degrade service slightly and have no effect on the comparison of Exchange or Scheduling algorithms.

Rather than a general treatment of all the Exchange methods discussed, a system simulation was conducted on a specific system.



LOAD AND QUIT OPERATIONS

FIGURE 10

This approach was followed to provide a better demonstration of the use to which the simulator could be put. Also, with the basic hardware capabilities fixed, the effectiveness of the various Exchange methods under differing operating conditions could be studied. The effect on performance of varying single system parameters was also investigated.

8.1.1 The basic simulated system configuration

The system investigated utilized a drum for the external storage device and did not permit concurrent transfers. An average access time of 15 milliseconds was used for all read and write transfer operations, and, as previously mentioned, a cycle time of three microseconds was used. Relocatability was provided and memory protection limited to 2K blocks except in special cases where the departure from this is noted. The normal core size was 32K, but this is a variable input parameter.

8.1.2 Exchange Method 1

The first Exchange Algorithm is the basic complete program exchange discussed in Section 5.3. The flow chart of Method 1 is provided in Figure 11a. As mentioned earlier, no program is transferred from core unless another user desires service. A check is first made to see if the program is in core and, if so, no transfer is required. If another user is in core, the normal dump, load and run sequence is implemented. The only memory protection required is the protection of the Executive Control Routine, and no relocatability is required.

Basic runs were made with 10, 20, 30, 40 and 50 users with a single job, with the job size increasing on each run and with a ten-job,

[illegible]

[illegible]

[illegible]

[illegible]

EXCHANGE METHOD 1

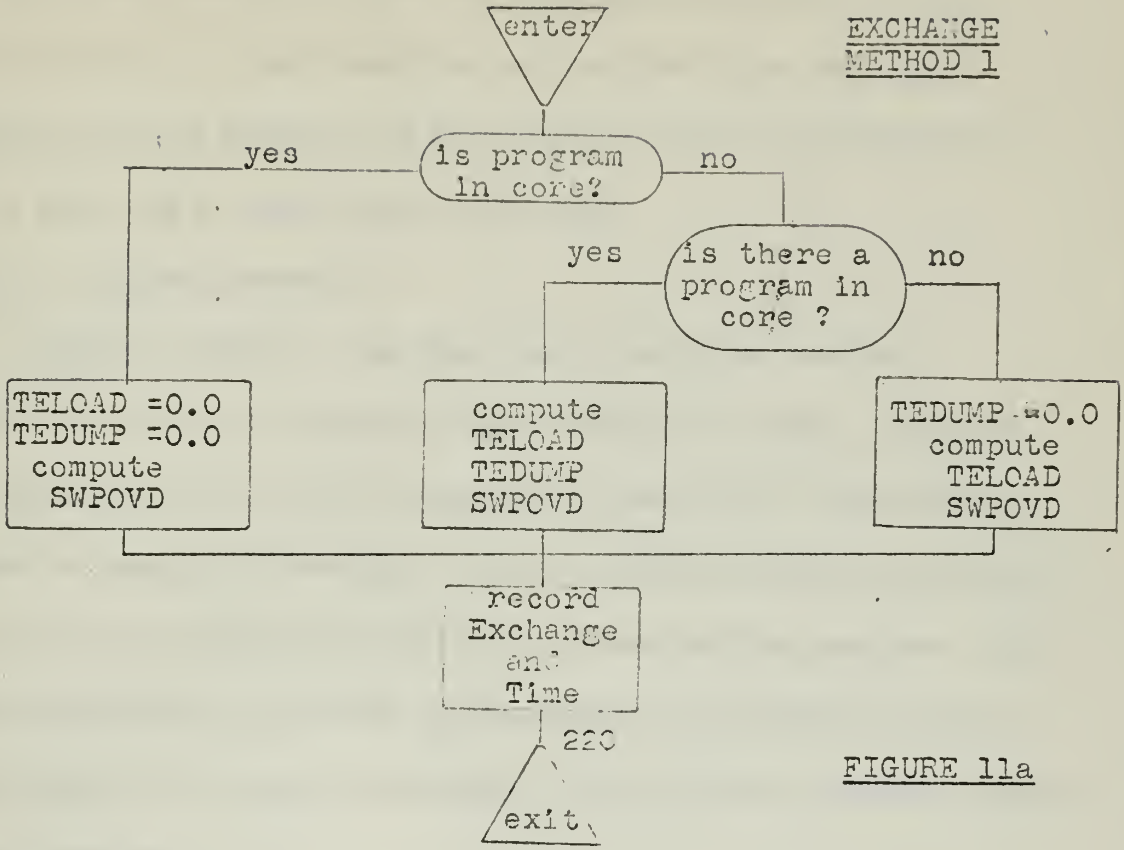


FIGURE 11a

EXCHANGE METHOD 2

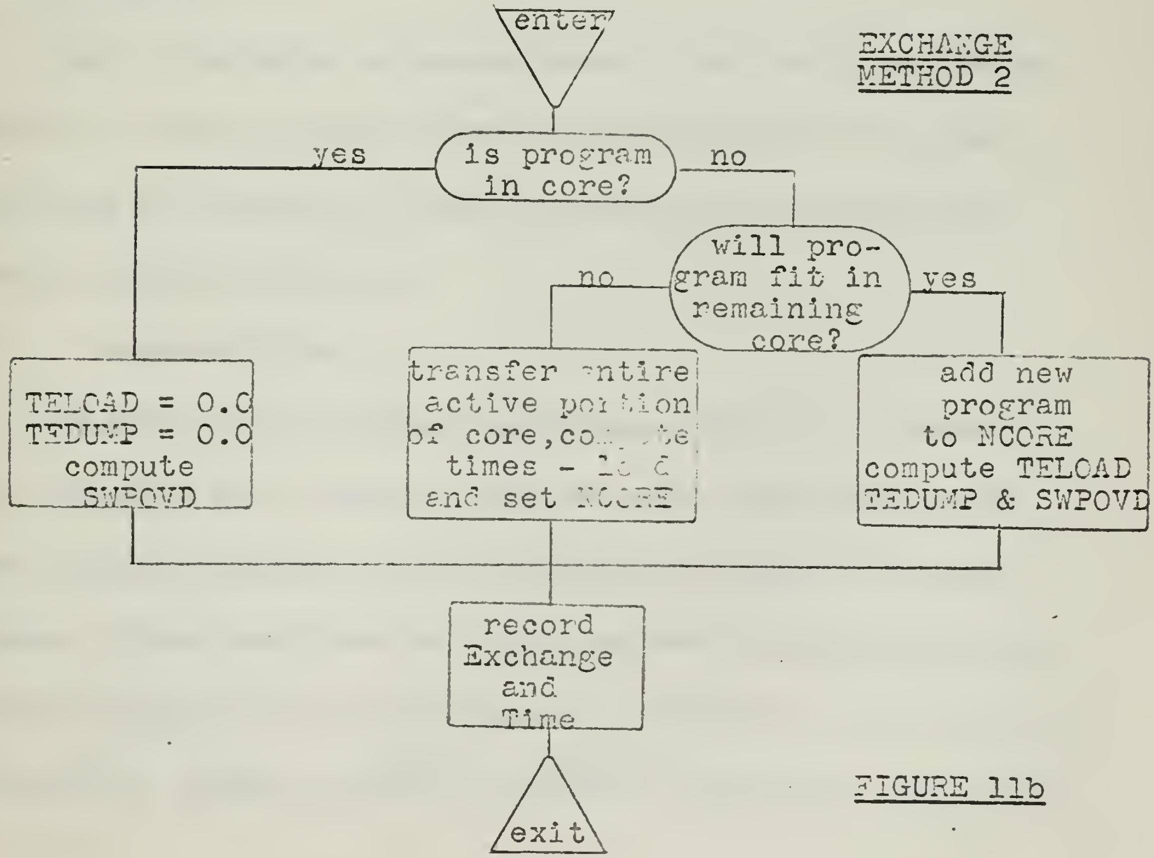


FIGURE 11b

widely mixed operating environment representing a typical computer center operation. The presentation and analysis of the data will be deferred until the latter part of this chapter to permit inclusion of all types and to allow comparisons to be made.

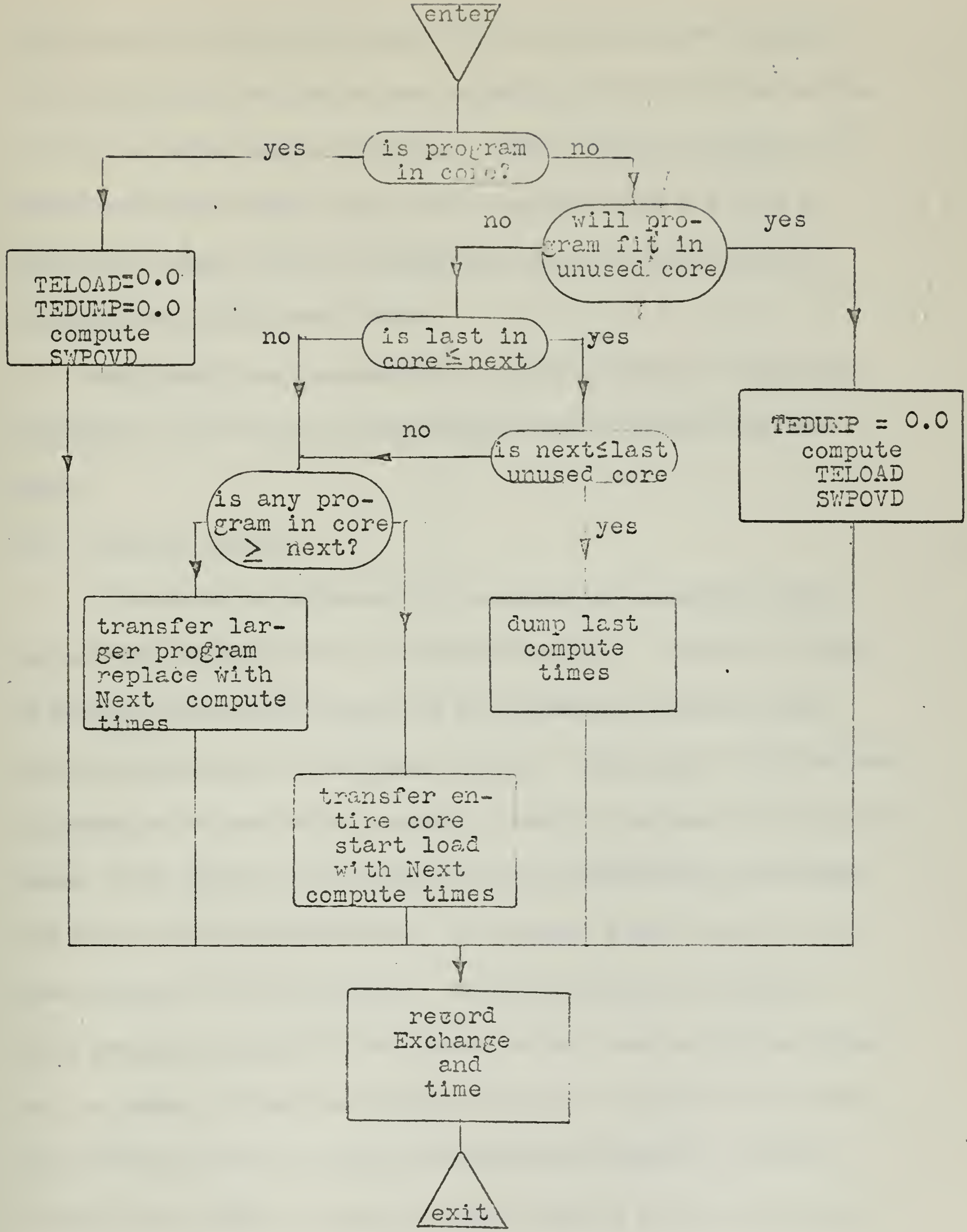
8.1.3 Exchange Method 2

Exchange Method 2, the flow chart of which is provided in Figure 11b, is a very rudimentary space allocation scheme. Programs are fitted into core one after another until core is full. At that time no selective dumping is attempted, and all programs in core are transferred to the external store and the build up restarted with the next user. Due to the requirement of checking memory bounds and preserving program environment of all programs transferred, the overhead is somewhat higher than for Method 1.

Runs corresponding to those of Method 1 were made. In addition separate runs were conducted using the mixed job input on a system with single cell resolution in memory protection and on another with protection limited to 2K blocks.

8.1.4 Exchange Method 3

The third and final Exchange algorithm, Method 3, is a complex space allocation type. Figure 12 provides a flow chart of the method. Core is filled sequentially until insufficient room exists for the next program. Several conditions are then tested with the aim of finding the smallest program that can be transferred to provide room, this minimizing transfer time. Memory protection is provided in 2K blocks. If no single



EXCHANGE METHOD 3

FIGURE 12

program can be transferred to allow the next user into core, including the combination of the last program sequentially plus the unused portion of core, the entire core is transferred. While various combinations of programs could be tested, the overhead required leads to a point of diminishing returns, and no combinations other than those using the unused portion of core were tested.

Again basic runs corresponding to those of Method 1 were made. In addition, a run with the mixed job load, and a 64K core size was made.

8.2 Simulator Output

To evaluate the performance of the Exchange algorithms tested, certain basic quantities were recorded for each run. The first of these is Exchange Time which consists of the summation of access times, actual transfer times and Exchange overhead. The second value recorded is Exchange Overhead which consists of specific overhead and all access times. Total Entries is the number of times the Exchanger was entered and an Exchange decision required. In Methods 2 and 3 Core Fits and Core Transfers are also recorded. Core Fits indicates the number of times programs were able to be inserted without transfer and Core Transfer, the number of times the complete core was transferred. The difference between the sum of these two and the Total Entries in Method 2 indicates the number of times the program required was already in core. Method 3 also records the number of Single Fits, or times a single program was transferred to make room for the new user. The second

line of output is generally self-explanatory. Number of Users is the number of stations in operation. Users Serviced provides a count of the number of users loaded into the system during the run. Jobs Generated actually provides an indication of jobs completed as fifty jobs are originally generated to start the run, and any number greater than this indicates jobs completed. Average Job Size is the average size of all jobs exchanged. No Loads and Load Wait Time are determined by drum size. No Loads is the number of times a LOAD request was received, but no room existed on the drum, and Load Wait Time is the average time a user waited to be loaded after receiving a No Load.

8.3 Simulation with Type 1 Inputs

The first series of runs conducted operated with a typical job mix that could be expected in a time-sharing type system. A mean arrival time of 300 seconds and a mean load time of one second were assumed. The other job characteristics are shown in Figure 13. The presence of several small, highly dynamic jobs with a background of larger compute limited jobs typifies a normal computation center loading.

Job Type	Active Time	I/O Time	Repeats	Mean Size	Job Probability
1	3.0	1.0	9.0	2000	0.150
2	1.0	3.0	15.0	4000	0.200
3	3.0	1.0	9.0	6000	0.200
4	2.0	2.0	30.0	8000	0.150
5	2.0	1.0	30.0	12000	0.100
6	1.0	2.0	15.0	16000	0.050
7	2.0	1.0	12.0	20000	0.050
8	2.0	2.0	15.0	24000	0.050
9	2.0	1.0	15.0	28000	0.020
10	1.0	1.0	15.0	32000	0.030

SIM JOB INPUT TABLE

FIGURE 13

A total of five different runs were made using this job mix as input. Each run consists of five individual one hour operating periods with 10, 20, 30, 40 and 50 users permitted respectively. The first three runs consisted of the three basic methods with a 32K core. Run 4 used Exchange Method 2 but assumed a single cell resolution for memory protection and transfers. Run 5 used Exchange Method 3 but allowed a 64K memory.

8.3.1 Analysis of results

The runs prove conclusively that as the mean core available per user becomes less than approximately thirty per cent of the mean job size, little advantage can be gained in the system under study, from more sophisticated space allocation Exchange techniques. When a large number of users are active, complete core transfers are required virtually every cycle, and the system behavior approaches that of the basic run and reload exchange regardless of the actual exchange method used.

The data from these runs is shown in Figures 14-18. The value of the single cell memory protection in this system appears to be marginal. The increased cost and timing problems created far outweigh the slight increase in performance. In comparing Figures 15 and 17 (block and single cell respectively) only a 1% increase in Total Entries and a 2.5% decrease in Exchange Time for the single cell configuration can be noted. Although increasing core (Figure 18) greatly increases exchange efficiency for the ten user system, by the time thirty users are allowed, the performance is only slightly improved over the 32K core Method 1 results (Figure 14).

8.4 Simulation with Type 2 Inputs

The second basic set of runs used a single job type, and, as the number of users was held constant, the mean program size was increased from 4K to 44K which resulted in average program size of approximately 32K. After each size increasing phase, the mean was reset to 4K, the number of users incremented and the run repeated. This set again emphasizes the relation between mean job size and mean size available per user with respect to Exchange method performance.

8.4.1 Analysis of results

The results are presented in Figures 19-21 which are further separated into a- e runs, each with a fixed number of users. As in the preceding simulation, with a mixed load, comparisons should not be made entirely on the basis of Exchange Time, but rather should concentrate on Total Entries which is a measure of the actual number of

computation quanta allowed during the operating period. The results using the Type 2 inputs are as expected. Matching Figures 19, 20 and 21 shows that the advantages of a particular method tend to disappear as the mean program size increases. As the number of users increases this crossover point occurs at a smaller mean size.

The simulation conducted shows the practical value of such a system design technique. Simulation provides valuable practical information on the relative value of hardware features such as memory and drum size and memory protection. Additionally, assistance is provided to the system programmer in the development of a suitable Executive Control Routine.

SIMULATION RESULTS

EXCHANGE METHOD ONE WITH A MIXED LOAD, TEN TO FIFTY USERS IN A SIMULATED ONE HOUR OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES
1372.539	463.725	15517.0
1322.344	493.616	16262.0
1390.910	490.698	16166.0
1399.685	492.858	16237.0
1387.215	491.288	16184.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
10.0	22.0	64.0	9753.9	.0	.0
20.0	26.0	58.0	8515.5	.0	.0
30.0	34.0	54.0	9303.3	.0	.0
40.0	42.0	52.0	9330.2	.0	.0
50.0	47.0	53.0	9248.5	337.0	1331.9

EXCHANGE METHOD 1

FIGURE 14

SIMULATION RESULTS

EXCHANGE METHOD TWO WITH A MIXED LOAD, TEN TO FIFTY USERS IN A SIMULATED ONE HOUR OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	CORE TRANSFERS
1360.385	459.226	15534.0	8955.0	6096.0
1340.654	490.540	16133.0	10730.0	5337.0
1406.911	486.366	16030.0	10260.0	5674.0
1418.415	489.609	16110.0	10147.0	5897.0
1407.225	489.117	16054.0	10179.0	5847.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
10.0	22.0	64.0	9615.4	.0	.0
20.0	26.0	59.0	8511.9	.0	.0
30.0	34.0	54.0	9307.2	.0	.0
40.0	42.0	52.0	9351.6	.0	.0
50.0	45.0	53.0	9253.8	336.0	656.9

EXCHANGE METHOD 2 WITH BLOCK PROTECTION

FIGURE 15

[illegible]

[illegible]

[illegible]

[illegible]

SIMULATION RESULTS

EXCHANGE METHOD THREE WITH A MIXED LOAD, TEN TO FIFTY USERS
IN A SIMULATED ONE HOUR OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	SINGLE TRANSFERS	CORE TRANSFERS
1372.354	464.120	15601.0	5717.0	5084.0	3525.0
1340.046	497.935	16126.0	5448.0	5926.0	3930.0
1410.962	496.451	15955.0	5667.0	4986.0	4118.0
1419.361	501.357	16031.0	5318.0	5599.0	4191.0
1407.361	499.667	16059.0	4782.0	6425.0	3967.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
10.0	22.0	64.0	9766.9	.0	.0
20.0	26.0	58.0	8505.5	.0	.0
30.0	34.0	54.0	9307.0	.0	.0
40.0	42.0	52.0	9334.0	.0	.0
50.0	44.0	52.0	9243.4	337.0	592.1

EXCHANGE METHOD 3 WITH 32K CORE

FIGURE 16

SIMULATION RESULTS

EXCHANGE METHOD TWO WITH A MIXED LOAD, TEN TO FIFTY USERS IN A SIMULATED ONE HOUR OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	CORE TRANSFERS
1336.124	456.472	15751.0	9016.0	5940.0
1314.184	491.661	16234.0	10863.0	5239.0
1388.732	491.031	16205.0	10492.0	5593.0
1397.466	493.284	16233.0	10312.0	5851.0
1389.850	493.628	16201.0	10353.0	5820.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
10.0	22.0	64.0	9604.9	.0	.0
20.0	26.0	59.0	8520.2	.0	.0
30.0	34.0	54.0	9303.1	.0	.0
40.0	42.0	52.0	9305.3	.0	.0
50.0	47.0	53.0	9245.0	337.0	1339.8

EXCHANGE METHOD 2 WITH SINGLE CELL RESOLUTION

FIGURE 17

SIMULATION RESULTS

EXCHANGE METHOD THREE WITH A MIXED LOAD, TEN TO FIFTY USERS
IN A SIMULATED ONE HOUR OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	SINGLE TRANSFERS	CORE TRANSFERS
872.293	305.584	19217.0	3051.0	4128.0	676.0
1291.921	472.299	16424.0	5813.0	5146.0	1156.0
1401.782	480.616	16059.0	7405.0	4369.0	1600.0
1411.124	483.247	16144.0	7402.0	4985.0	1455.0
1399.894	486.073	16114.0	8291.0	4398.0	1531.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
10.0	26.0	67.0	9650.2	.0	.0
20.0	27.0	59.0	8686.4	.0	.0
30.0	34.0	54.0	9424.7	.0	.0
40.0	42.0	52.0	9444.0	.0	.0
50.0	45.0	53.0	9260.6	336.0	654.6

EXCHANGE METHOD 3 WITH EXPANDED 64K CORE

FIGURE 18

[illegible]

[illegible]

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

[illegible]

SIMULATION RESULTS

EXCHANGE METHOD ONE WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES
76.078	43.735	1938.0
161.576	64.935	2422.0
192.165	60.007	2256.0
245.674	62.724	2384.0
334.136	71.281	2512.0
379.616	70.237	2445.0
375.502	62.658	2264.0
436.474	65.746	2349.0
473.802	63.974	2351.0
481.338	63.396	2331.0
485.550	67.666	2304.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
10.0	20.0	70.0	3827.4	.0	.0
10.0	19.0	67.0	7595.4	.0	.0
10.0	19.0	67.0	11304.5	.0	.0
10.0	19.0	68.0	14985.2	.0	.0
10.0	17.0	65.0	18746.3	.0	.0
10.0	17.0	64.0	22400.4	.0	.0
10.0	16.0	65.0	25562.2	.0	.0
10.0	15.0	63.0	28203.8	.0	.0
10.0	14.0	61.0	29830.4	.0	.0
10.0	14.0	59.0	30665.1	.0	.0
10.0	13.0	59.0	31343.6	.0	.0

FIGURE 19a

SIMULATION RESULTS

EXCHANGE METHOD ONE WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES
232.802	130.122	4300.0
301.232	115.654	3857.0
359.469	105.531	3524.0
402.198	95.316	3188.0
440.303	83.321	2925.0
474.734	81.695	2707.0
493.241	75.799	2513.0
507.090	73.154	2426.0
511.546	71.909	2385.0
513.104	71.240	2363.0
514.778	70.875	2351.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
20.0	27.0	66.0	3999.9	.0	.0
20.0	23.0	62.0	8020.4	.0	.0
20.0	23.0	58.0	12035.7	.0	.0
20.0	23.0	56.0	16098.9	.0	.0
20.0	21.0	55.0	20202.5	.0	.0
20.0	21.0	54.0	24390.5	43.0	210.7
20.0	19.0	54.0	27924.4	95.0	112.8
20.0	19.0	55.0	30081.4	114.0	93.1
20.0	18.0	55.0	30997.1	139.0	94.6
20.0	17.0	55.0	31442.7	147.0	117.2
20.0	17.0	55.0	31747.7	145.0	117.9

FIGURE 19b

SIMULATION RESULTS

EXCHANGE METHOD ONE WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES
233.347	129.819	4290.0
301.290	115.926	3833.0
356.711	104.799	3467.0
395.431	93.401	3092.0
436.477	86.865	2877.0
464.479	80.086	2654.0
492.543	75.861	2515.0
505.710	72.578	2407.0
513.506	71.514	2372.0
516.287	71.332	2366.0
515.611	70.967	2354.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
30.0	33.0	58.0	4042.3	.0	.0
30.0	33.0	56.0	8105.3	.0	.0
30.0	31.0	55.0	12185.0	.0	.0
30.0	30.0	55.0	16391.0	65.0	102.3
30.0	25.0	55.0	20403.9	95.0	94.8
30.0	22.0	55.0	24335.2	124.0	133.3
30.0	20.0	55.0	27853.4	140.0	85.3
30.0	19.0	55.0	30262.4	149.0	93.8
30.0	18.0	55.0	31332.4	153.0	77.9
30.0	17.0	55.0	31620.6	159.0	85.1
30.0	17.0	55.0	31759.8	158.0	85.2

FIGURE 19c

SIMULATION RESULTS

EXCHANGE METHOD ONE WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES
232.869	129.212	4270.0
299.694	114.802	3796.0
351.521	102.430	3389.0
398.057	93.796	3105.0
435.497	86.075	2851.0
467.572	80.391	2664.0
493.503	75.405	2500.0
510.568	72.699	2411.0
515.412	71.605	[illegible]375.0
515.614	71.210	[illegible]
517.516	71.210	[illegible]

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
40.0	41.0	56.0	4066.3	.0	.0
40.0	41.0	53.0	8163.3	.0	.0
40.0	36.0	53.0	12326.7	58.0	91.2
40.0	30.0	55.0	16443.3	80.0	189.2
40.0	25.0	55.0	20578.5	107.0	85.6
40.0	22.0	55.0	24417.8	130.0	105.3
40.0	19.0	55.0	28114.9	145.0	91.6
40.0	18.0	55.0	30538.9	152.0	89.9
40.0	18.0	55.0	31413.5	155.0	73.2
40.0	17.0	55.0	31635.2	160.0	80.0
40.0	17.0	55.0	31769.5	160.0	80.7

FIGURE 19[illegible]

SIMULATION RESULTS

EXCHANGE METHOD ONE WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES
231.889	129.030	4264.0
296.426	114.195	3776.0
355.984	103.494	3424.0
398.516	94.040	3113.0
430.978	85.167	2854.0
469.230	79.722	2642.0
488.011	75.436	2501.0
502.654	72.487	2404.0
507.696	71.089	2358.0
513.374	70.967	2354.0
513.167	70.633	2343.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
50.0	49.0	52.0	4040.0	.0	.0
50.0	48.0	51.0	8089.0	.0	.0
50.0	34.0	52.0	12365.5	67.0	75.7
50.0	29.0	54.0	16411.6	93.0	53.8
50.0	26.0	56.0	20286.4	115.0	80.4
50.0	22.0	56.0	24766.2	138.0	75.3
50.0	21.0	56.0	27733.9	147.0	60.9
50.0	20.0	56.0	30094.4	153.0	126.7
50.0	20.0	57.0	31136.9	160.0	116.4
50.0	19.0	57.0	31594.6	169.0	157.6
50.0	19.0	57.0	31758.8	163.0	157.8

FIGURE 19e

SIMULATION RESULTS

EXCHANGE METHOD TWO WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	CORE TRANSFERS
1.389	.824	1365.0	21.0	3.0
15.973	6.179	1634.0	133.0	64.0
181.895	56.100	2326.0	982.0	859.0
207.789	52.247	2133.0	743.0	974.0
334.220	71.073	2512.0	206.0	2136.0
379.672	70.001	2445.0	1.0	2308.0
375.925	62.424	2254.0	1.0	2059.0
436.443	66.533	2349.0	1.0	2193.0
473.790	68.771	2351.0	1.0	2267.0
481.326	68.194	2331.0	1.0	2248.0
485.540	67.465	2304.0	1.0	2224.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
10.0	21.0	71.0	3986.3	.0	.0
10.0	20.0	70.0	7752.0	.0	.0
10.0	19.0	69.0	11278.0	.0	.0
10.0	19.0	68.0	15025.2	.0	.0
10.0	17.0	65.0	18746.3	.0	.0
10.0	17.0	64.0	22400.4	.0	.0
10.0	16.0	65.0	25562.2	.0	.0
10.0	15.0	63.0	28203.8	.0	.0
10.0	14.0	61.0	29830.4	.0	.0
10.0	14.0	59.0	30663.1	.0	.0
10.0	13.0	59.0	31343.6	.0	.0

FIGURE 20a

SIMULATION RESULTS

EXCHANGE METHOD TWO WITH A SINGLE JOB TYPE OF VARYING MEAN
SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	CORE TRANSFERS
212.699	114.891	4359.0	3172.0	581.0
306.357	115.858	3851.0	2603.0	1191.0
360.251	105.201	3430.0	1784.0	1667.0
404.974	96.449	3191.0	929.0	2242.0
440.197	88.133	2925.0	101.0	2804.0
474.529	81.492	2707.0	1.0	2686.0
493.050	75.609	2513.0	1.0	2492.0
506.696	72.941	2425.0	1.0	2404.0
511.366	71.728	2335.0	1.0	2364.0
512.925	71.061	2363.0	1.0	2342.0
514.600	70.697	2351.0	1.0	2330.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
20.0	27.0	69.0	4004.3	.0	.0
20.0	23.0	61.0	8033.4	.0	.0
20.0	23.0	58.0	12057.1	.0	.0
20.0	23.0	56.0	16095.7	.0	.0
20.0	21.0	55.0	20202.5	.0	.0
20.0	21.0	54.0	24390.5	43.0	210.6
20.0	19.0	54.0	27924.4	96.0	112.7
20.0	19.0	55.0	30091.5	114.0	93.1
20.0	18.0	55.0	30997.1	139.0	94.6
20.0	17.0	55.0	31442.7	147.0	117.2
20.0	17.0	55.0	317[illegible]7.7	146.0	117.9

FIGURE 20b

SIMULATION RESULTS

EXCHANGE METHOD TWO WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	CORE TRANSFERS
232.012	123.884	4222.0	3403.0	648.0
303.781	114.066	3792.0	2539.0	1197.0
356.559	103.408	3422.0	1719.0	1674.0
398.150	93.538	3096.0	870.0	2206.0
436.305	86.662	2877.0	40.0	2817.0
464.278	79.887	2654.0	1.0	2633.0
492.352	75.672	2515.0	1.0	2494.0
505.527	72.396	2407.0	1.0	2386.0
513.327	71.335	2372.0	1.0	2351.0
516.108	71.153	2366.0	1.0	2345.0
515.433	70.789	2354.0	1.0	2333.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
30.0	34.0	58.0	4056.9	.0	.0
30.0	33.0	56.0	8098.2	.0	.0
30.0	31.0	55.0	12194.7	.0	.0
30.0	30.0	55.0	16394.5	67.0	104.1
30.0	25.0	55.0	20403.9	95.0	94.8
30.0	22.0	55.0	24335.2	124.0	133.3
30.0	20.0	55.0	27853.4	140.0	95.3
30.0	19.0	55.0	30262.4	149.0	93.8
30.0	18.0	55.0	31332.4	153.0	77.9
30.0	17.0	55.0	31620.6	159.0	84.6
30.0	17.0	55.0	31759.8	158.0	85.2

FIGURE 20c

SIMULATION RESULTS

EXCHANGE METHOD TWO WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	CORE TRANSFERS
233.684	124.188	4231.0	3403.0	657.0
303.307	113.437	3771.0	2512.0	1203.0
355.432	102.284	3385.0	1705.0	1651.0
397.396	93.140	3032.0	908.0	2154.0
435.319	85.872	2851.0	33.0	2798.0
467.371	80.191	2654.0	1.0	2643.0
493.313	75.217	2500.0	1.0	2479.0
510.385	72.517	2411.0	1.0	2390.0
515.137	71.426	2375.0	1.0	2354.0
515.435	71.031	2362.0	[illegible]	2341.0
517.338	71.031	2352.0	[illegible]	2341.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
40.0	42.0	56.0	4059.2	.0	.0
40.0	41.0	53.0	8161.2	.0	.0
40.0	36.0	53.0	12327.9	58.0	95.5
40.0	30.0	55.0	16446.9	79.0	192.8
40.0	25.0	55.0	20578.5	107.0	85.6
40.0	22.0	55.0	24417.8	130.0	105.7
40.0	19.0	55.0	28114.9	145.0	91.6
40.0	18.0	55.0	30538.9	152.0	89.9
40.0	18.0	55.0	31413.5	155.0	73.2
40.0	17.0	55.0	31635.2	160.0	80.0
40.0	17.0	55.0	31769.5	160.0	80.7

FIGURE [illegible]

SIMULATION RESULTS

EXCHANGE METHOD TWO WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	CORE TRANSFERS
236.390	126.049	4219.0	3462.0	661.0
301.732	113.571	3748.0	2529.0	1192.0
357.940	102.673	3394.0	1691.0	1678.0
397.441	93.315	3038.0	883.0	2125.0
430.828	85.971	2854.0	58.0	2776.0
469.030	79.524	2642.0	1.0	2621.0
487.822	75.247	2501.0	1.0	2480.0
502.472	72.306	2404.0	1.0	2383.0
507.517	70.910	2358.0	1.0	2337.0
513.100	70.789	2354.0	1.0	2333.0
512.990	70.456	2343.0	1.0	2322.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
50.0	49.0	52.0	4042.8	.0	.0
50.0	48.0	51.0	8090.0	.0	.0
50.0	34.0	52.0	12367.1	65.0	76.3
50.0	29.0	54.0	16416.5	92.0	53.9
50.0	26.0	56.0	20286.4	115.0	80.4
50.0	22.0	56.0	24766.2	138.0	75.3
50.0	21.0	56.0	27733.9	147.0	60.5
50.0	20.0	56.0	30094.4	158.0	126.7
50.0	20.0	57.0	31136.9	160.0	116.4
50.0	19.0	57.0	31594.6	169.0	157.5
50.0	19.0	57.0	31758.8	168.0	157.8

FIGURE 20e

SIMULATION RESULTS

EXCHANGE METHOD THREE WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	SINGLE TRANSFERS	CORE TRANSFERS
1.947	1.107	1355.0	17.0	12.0	3.0
102.988	40.781	2026.0	13.0	1323.0	4.0
162.069	50.781	2151.0	7.0	1657.0	4.0
211.856	55.038	2422.0	[illegible]	1777.0	19.0
335.427	71.561	2512.0	1.0	2137.0	[illegible]
351.099	66.335	2511.0	1.0	1680.0	469.0
375.787	65.171	2422.0	1.0	1100.0	950.0
440.615	67.418	2338.0	1.0	515.0	1691.0
435.213	63.651	2400.0	1.0	1742.0	347.0
477.893	67.858	2308.0	1.0	2228.0	1.0
483.472	67.390	2296.0	1.0	2213.0	1.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
10.0	21.0	71.0	3986.3	.0	.0
10.0	20.0	70.0	7680.3	.0	.0
10.0	19.0	69.0	11300.4	.0	.0
10.0	19.0	69.0	15111.6	.0	.0
10.0	17.0	65.0	18746.3	.0	.0
10.0	17.0	64.0	22663.6	.0	.0
10.0	16.0	65.0	25522.8	.0	.0
10.0	15.0	62.0	28207.3	.0	.0
10.0	15.0	64.0	29902.8	.0	.0
10.0	14.0	61.0	30692.2	.0	.0
10.0	14.0	59.0	31335.2	.0	.0

FIGURE 21a

SIMULATION RESULTS

EXCHANGE METHOD THREE WITH A SINGLE JOB TYPE OF VARYING MEAN
SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	SINGLE TRANSFERS	CORE TRANSFERS
163.586	92.974	4808.0	41.0	1365.0	6.0
282.873	108.881	3935.0	609.0	2434.0	202.0
340.893	100.649	3623.0	50.0	3211.0	24.0
390.401	94.354	3258.0	10.0	3074.0	9.0
438.390	88.356	2956.0	2.0	2898.0	2.0
474.827	81.788	2707.0	1.0	2686.0	.0
493.322	75.880	2513.0	1.0	2492.0	.0
506.950	73.193	2425.0	1.0	2404.0	.0
511.594	71.957	2335.0	1.0	2364.0	.0
513.148	71.283	2353.0	1.0	2342.0	.0
514.814	70.910	2351.0	1.0	2330.0	.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
20.0	27.0	70.0	3999.4	.0	.0
20.0	25.0	62.0	8010.1	.0	.0
20.0	23.0	58.0	12083.1	.0	.0
20.0	23.0	57.0	16098.0	.0	.0
20.0	21.0	55.0	20191.0	.0	.0
20.0	21.0	54.0	24390.5	43.0	210.7
20.0	19.0	54.0	27924.4	96.0	112.8
20.0	19.0	55.0	30081.5	114.0	93.1
20.0	18.0	55.0	30997.1	139.0	94.6
20.0	17.0	55.0	31442.7	147.0	117.2
20.0	17.0	55.0	31747.7	146.0	117.9

FIGURE 21b

SIMULATION RESULTS

EXCHANGE METHOD THREE WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	SINGLE TRANSFERS	CORE TRANSFERS
188.246	105.432	4574.0	169.0	1878.0	31.0
289.623	110.657	3808.0	663.0	2401.0	220.0
345.436	101.375	3495.0	158.0	3016.0	78.0
388.954	92.375	3165.0	7.0	3018.0	6.0
436.574	86.962	2877.0	1.0	2856.0	.0
464.570	80.177	2654.0	1.0	2633.0	.0
492.628	75.946	2515.0	1.0	2494.0	.0
503.778	72.645	2407.0	1.0	2386.0	.0
513.547	71.554	2372.0	1.0	2351.0	.0
516.325	71.370	2356.0	1.0	2345.0	.0
515.646	71.002	2354.0	1.0	2333.0	.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
30.0	34.0	59.0	4028.4	.0	.0
30.0	33.0	56.0	8090.6	.0	.0
30.0	31.0	56.0	12167.9	.0	.0
30.0	30.0	55.0	16386.6	65.0	78.8
30.0	25.0	55.0	20403.9	95.0	94.8
30.0	22.0	55.0	24335.2	124.0	133.4
30.0	20.0	55.0	27853.4	140.0	85.3
30.0	19.0	55.0	30262.4	149.0	93.8
30.0	18.0	55.0	31332.4	153.0	77.9
30.0	17.0	55.0	31620.6	150.0	85.1
30.0	17.0	55.0	31759.8	153.0	85.3

FIGURE 21c

SIMULATION RESULTS

EXCHANGE METHOD THREE WITH A SINGLE JOB TYPE OF VARYING MEAN
SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	SINGLE TRANSFERS	CORE TRANSFERS
198.864	110.141	4458.0	298.0	1825.0	56.0
277.364	106.386	3931.0	27.0	3443.0	8.0
345.165	100.400	3454.0	124.0	3051.0	61.0
388.270	91.910	3147.0	7.0	3003.0	6.0
435.598	86.176	2851.0	1.0	2830.0	.0
467.668	80.486	2654.0	1.0	2643.0	.0
493.584	75.486	2500.0	1.0	2479.0	.0
510.629	72.761	2411.0	1.0	2390.0	.0
515.357	71.646	2375.0	1.0	2354.0	.0
515.651	71.248	2352.0	1.0	2341.0	.0
517.551	71.245	2352.0	1.0	2341.0	.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
40.0	43.0	56.0	4054.6	.0	.0
40.0	41.0	54.0	8140.5	.0	.0
40.0	36.0	53.0	12322.2	58.0	91.1
40.0	30.0	55.0	16440.9	82.0	186.3
40.0	25.0	55.0	20578.5	107.0	85.6
40.0	22.0	55.0	24417.8	130.0	105.8
40.0	19.0	55.0	28114.9	145.0	91.6
40.0	18.0	55.0	30538.9	152.0	89.9
40.0	18.0	55.0	31413.5	155.0	73.2
40.0	17.0	55.0	31635.2	160.0	80.0
40.0	17.0	55.0	31769.5	160.0	80.7

FIGURE 21d

SIMULATION RESULTS

EXCHANGE METHOD THREE WITH A SINGLE JOB TYPE OF VARYING MEAN SIZE IN A REPRESENTATIVE FIFTEEN MINUTE OPERATING PERIOD

EXCHANGE TIME	EXCHANGE OVERHEAD	TOTAL ENTRIES	CORE FITS	SINGLE TRANSFERS	CORE TRANSFERS
207.026	115.545	4457.0	159.0	1840.0	27.0
287.593	110.489	3891.0	129.0	3271.0	42.0
347.098	101.358	3501.0	4.0	3324.0	1.0
388.159	92.655	3101.0	27.0	2977.0	26.0
431.074	86.263	2854.0	1.0	2833.0	.0
469.324	79.816	2642.0	1.0	2621.0	.0
488.092	75.516	2501.0	1.0	2480.0	.0
502.717	72.550	2404.0	1.0	2383.0	.0
507.745	71.139	2358.0	1.0	2337.0	.0
515.316	71.005	2354.0	1.0	2333.0	.0
513.202	70.668	2343.0	1.0	2322.0	.0

NUMBER OF USERS	USERS SERVICED	JOBS GENERATED	AVERAGE JOB SIZE	NO LOADS	LOAD WAIT TIME
50.0	49.0	53.0	4040.9	.0	.0
50.0	48.0	51.0	80[illegible]7.4	.0	.0
50.0	35.0	52.0	12316.5	67.0	84.0
50.0	29.0	54.0	16[illegible]9.8	96.0	52.5
50.0	26.0	56.0	202[illegible]6.4	115.0	80.4
50.0	22.0	56.0	24766.2	133.0	75.4
50.0	21.0	56.0	277[illegible]3.9	147.0	60.9
50.0	20.0	56.0	30094.4	158.0	126.7
50.0	20.0	57.0	31136.9	160.0	116.5
50.0	19.0	57.0	31594.6	169.0	157.6
50.0	19.0	57.0	31758.8	168.0	157.8

FIGURE 21e

9. Conclusions

This investigation provides a set of guidelines for the development of an Exchange technique for any system configuration. A general Exchange method can first be developed using heuristic analysis. Once the general approach applicable to the gross system has been determined a simulation, such as the type conducted on the sample system, will provide valuable information on the selection of the specific method and point out worthwhile areas of hardware improvement.

The investigation also points out the sensitivity of the Exchange techniques in general to small variations in the system configurations. It is impossible to define an optimal general Exchange method, as exchanges are dependent on both hardware limitations and operating environment. There are, however, six major parameters which, if defined, allow a general method to be selected. Minor parameters will modify the method slightly, but the approach provides a point of departure for the system designer. The parameters are:

1) <u>Mean Program Size</u> - This is expressed in relation to the core available to operating programs and can be roughly quantized into three primary levels of interest; small - less than one-third core, medium - approximately on-half available core and large - greater than half the available core.

2) <u>Core Size</u> - This is actual core available to operating programs, or the full core less than area required for the Executive. This can also be quantized as small - less than 32K, medium - 32 K to 64 K and large - above 64 K.

[illegible]

3) <u>External Store</u> - This is the type of external store available to the system. Typically, it consists of drum and/or disc, although any type of storage device such as tape or cartridge units is possible.

4) <u>Number of Simultaneous Transfer Channels</u> - This is the number of channels available for transfers and includes any necessary logical separation of memory to provide concurrent transfers.

5) <u>Number of Stations</u> - This is the maximum number of stations permitted to be in operation regardless of the type of service being received at a specific instant.

6) <u>Program Type</u> - This is a definition of the expected job mix and is defined in terms of compute or I/O limited jobs or jobs with large periods of man-machine communication.

Multiprogramming is a technique that will receive greater attention in the future, and the potential of such systems is virtually unlimited. This is best shown by a simple example. Imagine a system capable of handling one hundred stations at a time, and users who require only four hours of station operating time per day. In addition, the time-sharing period of only eight hours per working day would be required to provide half the rental cost of the system. An $8,000,000 system could be provided at a cost to the on-line users of $6.00 per hour. An increased number of stations or a longer operating period for time-sharing could reduce the cost still further. This hourly rate makes the power of such a system available to the myriad of small users who otherwise could not afford such a capability.

The full benefits of such a system cannot be realized without the best possible program exchange techniques. Although the hardware features will improve, the exchange method will still be required to do everything possible to compensate for the inherent slowness of transfer operations as compared to computation. In the next generation of systems, as in the present, the ability of a time-sharing system to meet its goal of more service to more users will be critically dependent on the performance of the Program Exchange Routine.

BIBLIOGRAPHY

1. Auerbach Corporation, Philadelphia, Pennsylvania. The Intent of the Opcon System Design; Master Control Philosophy, by L. J. Glodatien and B. H. Bragen. 19 July 1962. Technical Report 1069-TR-6.

2. Bauer, W. F. and W. L. Frank. DODDAC-An Integrated System for Data Processing, Interrogation and Display. Proceedings of the Eastern Joint Computer Conference 1961. The Macmillan Company 1961.

3. Bright, H. S. and B. F. Cheydleur. On the Reduction of Turnaround Time. Proceedings-Fall Joint Computer Conference 1962. Spartan Books, 1962.

4. Codd, E. F. Multiprogramming. Advances in Computers 1962 Academic Press. 1963.

5. Control Data Corporation. Programming Manual for Control Data Satellite Computer System. Pub. No. 187.

6. Control Data Corporation. Control Data 6600 Computer System Reference Manual. Pub. No. 450.

7. Corbato, F. J., M. Merwin-Daggett and R. C. Daley. An Experimental Time-Sharing System. Proceedings-Spring Joint Computer Conference 1962. Spartan Books, 1962.

8. Coyle, R. S. and J. K. Stewart. Design of a Real Time Programming System. Computers and Automation, September 1963.

9. Critchlow, A. J. Generalized Multiprocessing and Multiprogramming Systems. Proceedings - Fall Joint Computer Conference 1963. Spartan Books 1963.

10. Ferranti Electronics. The FP6000 Computer System.

11. Frank, W. L., W. H. Gardner and G. L. Stock. Programming On-Line Systems, Datamation, May-June 1963: 29-34, 28-32.

12. Fredkin, E. The Time-Sharing of Computers. Computers and Automation. November 1963: 12-20.

13. Gordon, G. A General Purpose Simulation Program. Proceedings of the Eastern Joint Computer Conference 1961. The Macmillan Company, 1961.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

14. Hogg, R. L. and D. C. Glover. Control System Programming Remote Computing and Data Display. U. S. Naval Postgraduate School M.S. Thesis 1963.

15. Hogg, R. L. and D. C. Glover. Program Control System for a Satellite Mode Computer Complex. Digital Control Laboratory, U. S. Naval Postgraduate School. 21 December 1962. Report TR-DCL-62-11/13.

16. Kelly, J. E., Jr. Techniques for Storage Allocation Algorithms. Communications of the ACM. October 1961: 449-453.

17. Kilburn, T., R. B. Payne and D. J. Howarth. The Atlas Supervisor. Proceedings of the Eastern Joint Computer Conference, 1961. The Macmillan Company 1961.

18. Kilburn, T., D. J. Howarth, R. B. Payne and F. H. Sumner. The Manchester University Atlas Operating System. The Computer Journal, October 1961: 3-10.

19. Lewis, J. W. Time Sharing on Leo III. The Computer Journal, April 1963: 24-28.

20. Maher, R. J. Problems of Storage Allocation in a Multiprocessor Multiprogrammed System. Communications of the ACM. October 1961: 421-422.

21. Markowitz, H. M., B. Hauser and H. W. Kau. Simscript - A simulation Programming Language. Prentice-Hall, 1963.

22. McKenney, J. L. Simultaneous Multiprogramming of Electronic Computers. University of California, Los Angeles Doctoral Dissertation. February 1961.

23. Mills, M. R. Operational Experience of Time Sharing and Parallel Processing. The Computer Journal, April 1963: 28-36.

24. Riskin, B. N. Core Allocation Based on Probability. Communications of the ACM, October 1961: 454-459.

25. Ryle, B. L. Multiple Programming Data Processing. Communications of the ACM, February 1961: 99-101.

26. Shafritz, A. B., A. E. Miller and K. Rose. Mulrilevel Programming for a Real-Time System. Proceedings of the Eastern Joint Computer Conference, 1961. The Macmillan Company, 1961.

27. Smith, R. D. Multiprogramming the RCA 601. Preprints of Papers Presented at the 16th National Meeting, Association For Computing Machinery. 1961.

28. Statland, N. and J R. Hillegass. Random Access Storage Devices. Datamation, December 1963.

29. System Development Corporation, Command Research Laboratory User's Guide, 7 November 1963. Report TM-1354 volumes 1-5

30. Weil, J. W. A Heuristic for Page Turning In a Multiprogrammed Computer. Communications of the ACM, September, 1962: 480-481.

31. Wilder, W. G. An Investigation of the Scheduling Aspects of Multiprogramming, U. S. Naval Postgraduate School M.S. Thesis 1964.

32. Williams, R. J., J. P. Anderson, S. A. Hoffman and J. Shifman. D825- A Multiple-Computer System for Command and Control. Proceedings-Fall Joint Computer Conference 1962. Spartan Books, 1962.

33. Yarborough, L. D. Some Thoughts on Parallel Processing. Communications of the ACM, October, 1960: 30-31.

APPENDIX I

SIM-A MULTIPROGRAMMING SYSTEM SIMULATOR

Simulation has become a valuable analytic method that can be applied in diverse situations. The type of simulation of interest is the Monte Carlo Method which is defined in the McGraw Hill Encyclopedia of Science and Technology, as "A technique for estimating the solution, x, of a numerical mathematical problem by means of an artifical sampling experiment....." The method aptly fits the multiprogramming system problem and can produce worthwhile results. The required probability distributions associated with users can be determined by general data gathering and observation. The use of various algorithms in the Executive routine and several hardware configurations in a simulated system subjected to a typical loading will produce the data to obtain a measure of effectiveness for the various hardware and software configurations. The time and expense required to actually evaluate each of the combinations in an operating system are prohibitive and the use of simulation techniques provides the only realistic approach.

Program SIM was developed as a general multiprogramming system simulator with the emphasis on the time-sharing type of environment. Due to the specific nature of the authors' theses, primary attention was given to the Scheduler and Exchanger. The normal performance of other specific areas of the Executive routine is assumed and these portions treated in a block method. A prime example of this is the Dispatcher. While it is a critical area of a multiprogramming

[illegible]

[illegible]

[illegible]

[illegible]

system no specific characteristics were assumed. When a user has an Input or Output the program assumes a waiting status until, due to the incrementing of the simulator clock, the action is deemed complete. The availability of the required I/O equipment at all times is assumed. A time-sharing system is characterized by frequent man-machine communication and buffered I/O is usually impossible due to the step by step nature of the system. However, simultaneous I/O by all users, at least to their reactive typewriters, must be permitted. The gross Dispatcher treatment provides all this and only avoids the complications of particular operations. If this area is studied in the future the Dispatcher portion could easily be made more detailed and added to the system simulator.

The job load on the simulated system is created by a job generation subroutine (SET). Each job is characterized by six variables, which define any job entering the system. Arrival time, the first parameter, is assumed to be exponentially distributed on the basis of queuing theory concepts and actual observation at System Development Corporation. A variable parameter is the mean arrival time expressed in seconds. The value of arrival time was determined by taking the natural logarithm of a uniformly distributed random number. The second parameter is Load time and represents the time required to transfer the binary program from its permanent storage to the temporary storage having access to the central memory. The next three parameters, Active time, I/O time and Repeats define the actual program operating

characteristics. A program once loaded into the system is assumed to have an active period followed by an I/0 period, during which no service is required from the central processor. This cycle is repeated until the job is completed or there are no further repeats required. Due to the nature of SIM, differences in I/0 such as tape transfers, searches or outputs to reactive devices are not recognized and the I/0 operations are grouped together. The sixth parameter, Size, completes the job description. Program size is limited to a maximum length of one hundred cells less than the full core available for operating programs. The last five parameters are determined using a Gaussian Random number generator and the mean values received as input. A uniform random number generater is used to generate any of ten possible job types. The probability of each type job is received as an input. As soon as the job is completed, that is the number of repeats remaining is less than zero, a new job is generated for that station. An example of the input to SIM is shown at the end of the program contained in Figure I-1.

It is possible to obtain a wide variety of output parameters from the simulator as it has access to all of the internal system parameters concerning the operation of the system in question. These parameters may be gathered on a minute, average, total or maximum basis and thereby present a picture of the system's operation in almost any degree of detail desired.

For the purpose of comparison, the output parameters of one run, using a certain hardware and software configuration, may be saved and then presented with the results of another run with changes to the hardware and/or software configuration. The output of the simulator may be in a tabular format or modified to a graphical type format, which ever is deemed best for comparison purposes.

The program operation is cyclic in nature. First, the initial jobs are generated and the program constants read in and/or initialized for the run. The main body of the simulator is then entered and the actual run commenced. The clock is checked against arrival times of all allowed users (maximum 50) and an equality or greater than condition sets the action entries of the status table (STAT(X,Y)). The Scheduler then determines which requests for service shall be honored and the order in which they shall be honored, i.e., queue information. The Scheduler also determines, from the number requesting service, the amount of time each user is allowed per cycle. The cycle begins with the formation of the queue and ends with termination of the last user's quantum. The Scheduler then passes control to the Exchanger. For further specific information concerning the operation of the Scheduler section of the simulator see Lt. W. G. Wilder's thesis.

The Exchanger determines the action required by the next user in the queue and then LOADS, QUITS or TRANSFERS the users program. The actual transfer algorithm is variable and the methods used are discussed in detail in Lt. R. R. Hatch's thesis. Regardless of the

[illegible]

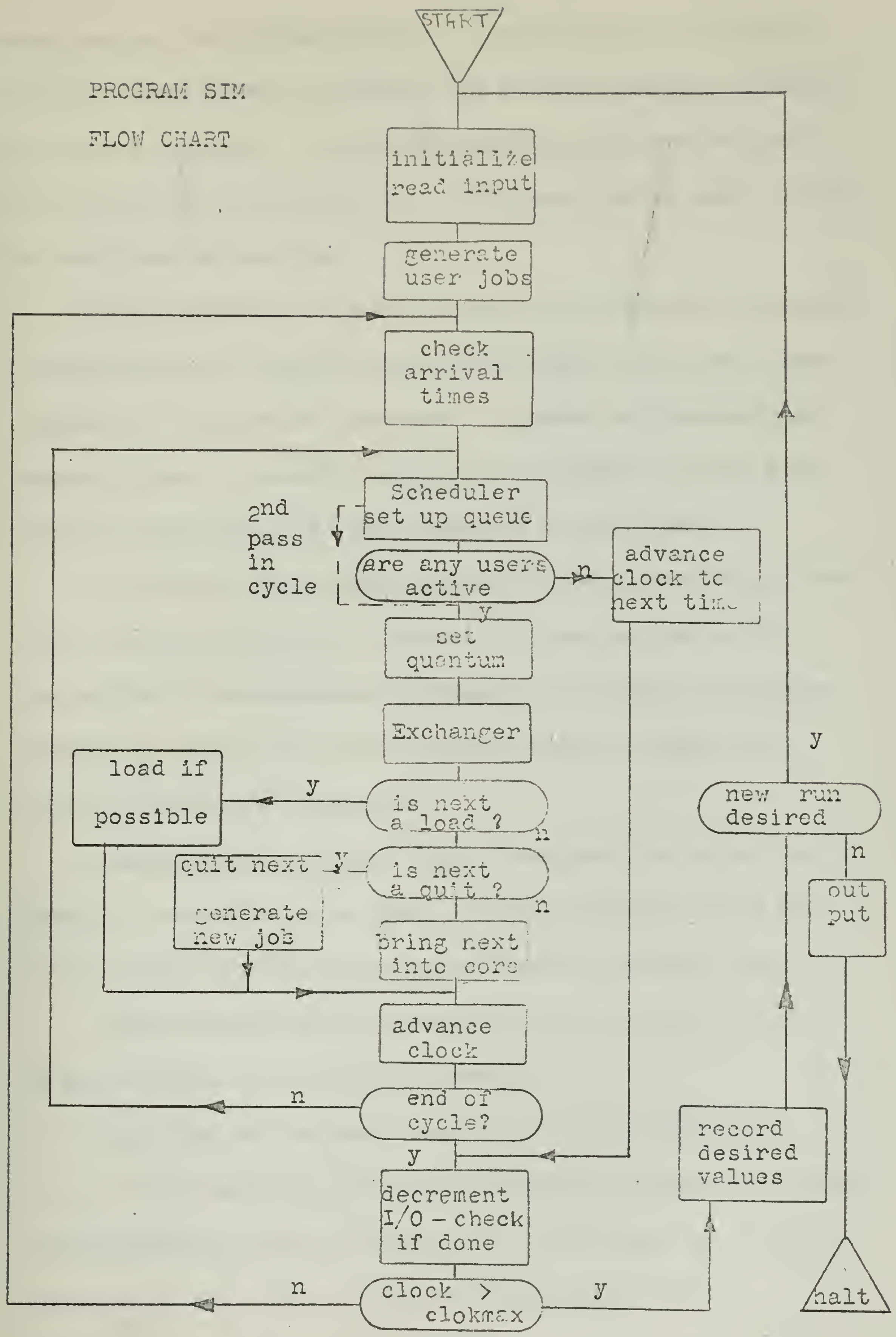

START
PROGRAM SIM
FLOW CHART
initialize read input
generate user jobs
check arrival times
Scheduler set up queue
2nd pass in cycle
are any users active
n
advance clock to next time
y
set quantum
Exchanger
load if possible
y
is next a load ?
n
quit next generate new job
y
is next a quit ?
n
bring next into core
advance clock
n
end of cycle?
y
decrement I/O - check if done
n
clock > clokmax
y
record desired values
new run desired
y
n
out put
halt

FIGURE I-1

exact method, the required transfers are determined and the effective transfer times (TELOAD and TEDUMP) and exchange overhead calculated and added to the clock. In the LOAD and QUIT operations the size of the external store, such as a drum, is considered and no load or storage full conditions are possible.

At the completion of a cycle all users in an I/0 status are handled by decrementing the remaining time by the elapsed cycle time. Users completing I/0 are checked for repeats. If repeats are necessary the program is reset to the active mode and if no repeats are required the program is terminated by a QUIT command in the next cycle.

To avoid long idle periods a scheme is used to advance the clock to the next active clock time if there are no users desiring service. The smallest I/0 time remaining (SMALLA) and the nearest arrival time (SMALLB) are determined and the smaller of these two added to the clock and a new cycle commenced.

A maximum clock parameter read in terminates the run and the capability for recycling is provided. All new parameters may be read in or the original parameters may be modified for successive runs.

A flow diagram of SIM is contained in Figure I-2 and a copy of the actual program is contained in Appendix II.

Due to fact that the thesis topics of Lt. W. G. Wilder and Lt. R. R. Hatch are both in the multiprogramming area and a generalized multiprogramming simulator could be used in each case, this simulator represents the joint efforts of both Lt. Wilder and Lt. Hatch.

APPENDIX II

PROGRAM SIM

```
      DIMENSION SAVE(50,5)
      DIMENSION MCORE(50,4) , MDRUM(50,4) , MDISC(50,4)
      DIMENSION IQUIT(50),IQUE(50),GENR(50,10),STAT(50,15),
     1AVERAGE(10,8),OUTABL(15),INPUT(40),DIST(10),DATA(10),TOTOUT(30,50)
      COMMON GENR,AVERAGE,DIST,JN,JOBTYPE,CLOCK,DST,IOUTCON
      COMMON MAXDRM,MAXDSC,MAXCOR,IR
      COMMON GAUSRN, UNIRN
    1 FORMAT(7F10.3)
    6 FORMAT (7F10.3)
    7 FORMAT(6I2)
    9 FORMAT(7F10.3)
   50 FORMAT(4I6)
   51 FORMAT(1H1)
   52 FORMAT(////)
   54 FORMAT(30X,32HWITH EARLY TERMINATIONS ALLOWED,)
   55 FORMAT(25X,41HNOT ALLOWED AND BACKGROUND JOBS PERMITTED//)
   60 FORMAT(/////////)
   80 FORMAT(//2X,8HVARIABLE,5X,15HCYCLE    AVERAGE,3X,7HAVERAGE,3X,
     17HAVERAGE,3X,21HAVERAGE COMPUTATIONAL,8H AVERAGE,2X,7HAVERAGE)
   81 FORMAT(10H PARAMETER,5X,14HCOUNT     CYCLE,4X,6HNUMBER,4X,7HQUANTUM
     1,2X,19HOVERHEAD EFFICIENCY,3X,8HEXCHANGE,2X,8HEXCHANGE)
   82 FORMAT(24X,4HTIME,4X,8HIN QUEUE,34X,8HOVERHEAD,3X,4HTIME//)
   83 FORMAT(//2X,7HAVERAGE,4X,7HMAXIMUM,3X,7HMAXIMUM,2X,7HMAXIMUM,3X,
     17HMAXIMUM,3X,7HMAXIMUM,3X,7HMAXIMUM,3X,8HSTATIONS)
   84 FORMAT(2X,8HOVERLOAD,3X,7HQUANTUM,3X,5HCYCLE,4X,6HNUMBER,4X,
     18HOVERHEAD,2X,8HOVERLOAD,2X,6HNUMBER,4X,8HSERVICED)
   85 FORMAT(23X,4HTIME,5X,8HIN QUEUE,22X,8HSTATIONS//)
   86 FORMAT(//2X,3HJOB,4X,20HCLOCK    STATION  JOB,4X,7HARRIVAL,3X,
     14HLOAD,6X,6HACTIVE,9X,3HI/C,5X,7HREPEATS,5X,4HSIZE)
   87 FORMAT(6HNUMBER,3X,4HTIME,4X,12HNUMBER   TYPE,4X,4HTIME,11X,4HTIME
     16X,4HTIME,10X,4HTIME//)
   88 FORMAT(//40X,8HFIGURE 7//)
   89 FORMAT(35X,21HSIMULATOR OUTPUT DATA)
   90 FORMAT(2F15.3,F10.6,3F15.1)
   91 FORMAT(F15.1,2F6.1,F12.6,2F9.1)
   92 FORMAT(10F9.6)
```

```
   93 FORMAT(//44X,8HFIGURE 6//)
   94 FORMAT(39X,18HGENERATED JOB DATA////)
  515 FORMAT(9F10.4//)
 516  FORMAT(8F10.4//)

      IR=1
      ID=0
      IE=0
      NREADF=0
      PRINT 51
      PRINT 60
      PRINT 93
      PRINT 94
      PRINT 86
      PRINT 87
      READ 1,((AVERAGE(J,I),I=1,7 ),J=1,10)
C     INITIALIZE
   10 ENDCYCL=1.0
      JN = 0
      JP=0
      JD=0
      JC=0
      UNIRN = 0.374821
      INP=0
      IF(MEHLT)15,12,15
   12 CONTINUE
      GO TO 14
   16 CONTINUE
      ISTOPF = 1
      GO TO 570
   15 CONTINUE
      PLENGTH = AVERAGE(I,6)
      AVERAGE(I,6)=PLENGTH + 1000.0
      AVERAGE(1,7) = 1.0
      IF(AVERAGE(1,6) - 64000.0) 14,14,16
   14 CONTINUE
      IF(NREADF)45,45,46
   45 CONTINUE
      READ 50, MAXDRM,MAXDSC, MAXCOR ,MEHLT
      READ 6,QA,QB,QC,QD,QE,QF,SOVRHD1
      READ 9,SOVRHD2,SOVRHD3,CLOKMAX,EMINQ
      READ 7,NCYCTM,ISTOPF,ISKEDTP,NUSERS,IVARY,IOUTCON
      NU=NUSERS
   46 CONTINUE
```

```
      JOBKIND=4
      DST=0.0
      CLOCK=0.0
      DO 3 J=1,JOBKIND
    3 DIST(J)=0.0
      DO 4 J=1,50
      DO 4 I=1,10
      GENR(J,I)=-1.0
    4 STAT(J,I)=-1.0
C     GENERATE FIRST 50 JOBS
      DO 5 J=1,50
    5 CALL SET(J)
      NDRUM=0
      SOVD = 0.0
      EXOVO = 0.000030
      EXOVF = 0.000100
      LDCHK=0
      NWCYCL=1
      DRUM=0.0
      CORET = 0.0
      WAITT = 0.0
      WAITC = 0.0
      CYCNT=0.0
      COREUT=0.0
      QTOT=0.0
      TCYTM=0.0
      TNQUE=0.0
      NOLOAD = 0
      TEFEXCH =0.0
      TEXCH = 0.0
      CSOVRHD=0.0
      SOVRHD=0.0
      SOVRHDM=0.0
      CYTIMMX=0.0
      MNQUE=0
      MNOVLD=0
      ITOVLD=0
      INU=1
      REQUEST=0.0
C     SCHEDULER SET-UP
      GO TO (20,21,22,23),IVARY
   20 VARPARA=MINQ
      GO TO 25
```

```
   21 VARPARA=ISKEDTP
      GO TO 25
   22 VARPARA=NCYCTM
      GO TO 25
   23 VARPARA=NUSERS
   25 CONTINUE
      FCYCTM=NCYCTM
      IQMAX=FCYCTM/EMINQ
      GO TO (30,31,32,33,34),ISKEDTP
C     REGULAR ALGORITHM
   30 INET=2
      GO TO 100
C     EARLY TERM. NOT ALLOWED
   31 INET=1
      GO TO 100
C     BACKGROUND USERS ALLOWED
   32 GENR(1,1)=CLOCK
      GENR(1,3)=CLOKMAX
      INU=2
      INET=2
      GO TO 100
C     ARRIVAL TIME AVERAGE VARIABLE
   33 DO 40 IB=1,10
      INET=2
      AVE=AVERAGE(IB,1)
      AVERAGE(IB,1)=AVE-50.0
      IF(AVERAGE(IB,1))41,41,40
   40 CONTINUE
      NREADF=1
      ID=ID+1
      IF(ID-10)42,42,41
   41 ISTOPF=1
      NREADF=0
      GO TO 570
   42 GO TO 100
C     NUMBER OF USERS VARIABLE
   34 NU=NU+5
```

```
      INET=2
      IF(IVARY-4)48,49,48
   49 VARPARA=NU
   48 CONTINUE
      NREADF=1
      IE=IE+1
      IF(IE-10)44,43,43
   43 NREADF=0
   44 GO TO 100

C     ARRIVAL DETERMINATION

  100 TIMERUN=0.0
      I=1
  101 IF(GENR(I,1)-CLOCK) 102,102,103
  102 STAT(I,1)=1.0
      STAT(I,2)=1.0
      STAT(I,9)=1.0
      GEN=GENR(I,1)
      GENR(I,1)=GEN+1000000.0
  103 IF(I-NU) 104,105,105
  104 I=I+1
      GO TO 101
  105 SCYCLE=CLOCK
      GO TO 1000

C     NO ACTIVE USERS

  400 CONTINUE
      ENDCYCL=1.0
      CYTIME=CLOCK-SCYCLE
      SMALLB = GENR(1,1) - CLOCK
      DO 401  I =2,NU
      IF(GENR(I,1) - CLOCK) 401,410,410
  410 CONTINUE
      IF(GENR(I,1) - CLOCK - SMALLB) 402,401,401
  402 SMALLB = GENR(I,1) - CLOCK
  401 CONTINUE
      SMALLA = STAT(1,5)
      IF(SMALLA) 403,404,404
  403 SMALLA = 1000000.0
  404 CONTINUE
      DO 407 I=2,NU
      IF(STAT(I,5)) 407,408,408
  408 CONTINUE
      IF(SMALLA - STAT(I,5)) 407,407,409
  409 SMALLA = STAT(I,5)
```

```
  407 CONTINUE
      IF(SMALLA-SMALLB)405,406,406
  405 CLOCK=CLOCK+SMALLA
      GO TO 360
  406 CLOCK=CLOCK+SMALLB
      GO TO 360

C     USER IS LOADING

  200 N=IQUE(NEXT)
      REQUEST=REQUEST+1.0
      LDCHK=0
      CLOCK=CLOCK+GENR(N,2)
      STAT(N,4)=GENR(N,3)
      GO TO 300

C     USER IS QUITTING

  210 I=IQUE(NEXT)
      STAT(I,1)=-1.0
      CALL SET(I)
      GO TO 300

C     USER IS ACTIVE

  220 N=IQUE(NEXT)
      IF(STAT(N,4)-Q)221,221,222
  222 ACTIME=STAT(N,4)-Q
      STAT(N,4)=ACTIME
      TIMERUN=Q
      CLOCK=CLOCK+Q
      GO TO 300

C     EARLY TERMINATION

  221 TIMERUN=STAT(N,4)
      TERM=1.0
      STAT(N,9)=-1.0
      STAT(N,5)=GENR(N,4)
      CLOCK=CLOCK+TIMERUN
      GO TO 300

C     END OF CYCLE CHECK AND HOUSE-KEEPING

  300 IF(ENDCYCL)1000,1000,350
  350 CYTIME=CLOCK-SCYCLE
 360  CONTINUE
```

```
      ACYTIME=CLOCK-SCYCLE
      DO 351 I=1,NU
      IF(STAT(I,5))351,351,352
  352 IF(STAT(I,10))358,358,357
  358 STAT(I,10)=1.0
      GO TO 351
  357 TIME=STAT(I,5)-ACYTIME
      IF(TIME)353,353,354
  354 STAT(I,5)=TIME
      GO TO 351
  353 CONTINUE
      STAT(I,5)=-1.0
      STAT(I,10)=-1.0
      IF(GENR(I,5))356,356,355
  355 STAT(I,4)=GENR(I,3)
      STAT(I,9)=1.0
      REPT=GENR(I,5)-1.0
      GENR(I,5)=REPT
      GO TO 351
  356 CONTINUE
      STAT(I,9)=1.0
      STAT(I,6) = 1.0
  351 CONTINUE
      IF(CLOCK-CLOKMAX) 100,100,500

C BASIC SCHEDULER

 1000 IF(ENDCYCL)1060,1060,1050
 1060 IF(TERM)1020,1020,1430
 1430 GO TO (1020,1040),INET

C     NEW CYCLE

 1050 IT = 1
      ENDCYCL=0.0
      IQ = 1
      NEXT=0
      CLOCK=CLOCK+SOVRHD3
      SOVRHD=SOVRHD+SOVRHD3

C     QUEUE FORMATION FIRST PHASE

      DO 1120 IS=INU,NU
      IF(STAT(IS,9))1120,1120,1070
 1070 IF(STAT(IS,2))1010,1010,1080
 1080 IF(LOCHK)1090,1090,1120
 1090 ILOD =IS
```

```
      LDCHK=1
      GO TO 1120
 1010 IF(STAT(IS,6))1100,1100,1110
 1110 IQUIT(IT)=IS
      IT=IT+1
      GO TO 1120
 1100 IQUE(IQ)=IS
      IQ=IQ+1
      IF(IQ-LQMAX)1120,1120,1440
 1120 CONTINUE
      IF(ISKEDTP-3)1441,1442,1441

C      BACKGROUND USERS ALLOWED

 1442 IQUE(IQ)=1
      IQ=IQ+1
      GO TO 1040
 1441 CONTINUE
 1440 IF(LDCHK)1040,1400,1040
 1400 IF(IQ-1)1040,1121,1040
 1121 IF(IT-1)1040,400,1040

C     1040 FOR ANY REQUESTS 400 FOR NC REQUESTS

 1040 IQ=IQ-1

C     STATISTICS GATHERING

      CLOCK=CLOCK+SOVRHD2
      SOVRHD=SOVRHD+SOVRHD2
      NQUE=IQ
      NOQ=0
      DO 1570 I=1,NU
      IF(STAT(I,9))1570,1570,1250
 1250 NOQ=NOQ+1
 1570 CONTINUE
      NOVLD=NOQ-NQUE-IT+1-LDCHK
      ITOVLD=ITOVLD+NOVLD
      TOVLD=ITOVLD
      AOVLD=TOVLD/CYCNT
      IF(MNOVLD-NOVLD)1380,1380,1390
 1380 MNOVLD=NOVLD
 1390 FMNOVLD=MNOVLD

C     EARLY TERMINATION

      IF(TERM)1530,1530,1540
```

```
 1540 FNQUE=NQUE-NEXT+1
      FNCYCTM=NCYCTM
      FNEXT=NEXT
      FNCYCTM=FNCYCTM-QAVE*FNEXT
      Q=FNCYCTM/FNQUE
      TERM=-1.0
      GO TO 1550
C     QUANTUM DETERMINATION FIRST PHASE
 1530 JQ=NCYCTM/NQUE
      Q=JQ
 1550 IF(IT-1)1500,1500,1030
C     QUEUE FORMATION SECOND PHASE
 1030 DO 1130 I=1,IT
 1130 IQUE(NQUE+IT-1)=IQUIT(IT-1)
 1500 IF(LDCHK)1520,1520,1510
 1510 IQUE(NQUE+IT)=ILOD
 1520 CONTINUE
      NWCYCL=1
C     QUANTUM DETERMINATION SECOND PHASE
      IF(Q-QA)1140,1150,1150
 1150 Q=QA
      GO TO 1020
 1140 IF(Q-QB)1160,1170,1170
 1170 Q=QB
      GO TO 1020
 1160 IF(Q-QC)1180,1190,1190
 1190 Q=QC
      GO TO 1020
 1180 IF(Q-QD)1200,1210,1210
 1210 Q=QD
      GO TO 1020
 1200 IF(Q-QE)1220,1230,1230
 1230 Q=QE
      GO TO 1020
 1220 Q=QF
      GO TO 1020
C     OLD CYCLE
 1020 NEXT=NEXT+1
      TERM=-1.0
```

```
      IF(NWCYCL)1410,1420,1410
C     STATISTICS GATHERING
 1410 CYCNT=CYCNT+1.0
      QTOT=QTOT+Q
      QAVE=QTOT/CYCNT
      TCYTM=TCYTM+CYTIME
      CYTMAVE=TCYTM/CYCNT
      FNQUE=NQUE
      TNQUE=TNQUE+FNQUE
      AVNQUE=TNQUE/CYCNT
      CSOVRHD=CSOVRHD+SOVRHD
      AOVRHD=CSOVRHD/CYCNT
      COMPEFF=(AVNQUE*QAVE/CYTMAVE)*100.0
      IF(QMAX-Q)1300,1300,1310
 1300 QMAX=Q
 1310 IF(CYTIMMX-CYTIME)1320,1320,1330
 1320 CYTIMMX=CYTIME
 1330 IF(MNQUE-NQUE)1340,1340,1350
 1340 MNQUE=NQUE
 1350 IF(SOVRHDM-SOVRHD)1360,1360,1370
 1360 SOVRHDM=SOVRHD
 1370 FMNQUE=MNQUE
 1420 CLOCK=CLOCK+SOVRHD1
      NWCYCL=0
      SOVRHD=SOVRHD+SOVRHD1
C     END OF CYCLE DETERMINATION
      IF(NEXT-NQUE-IT+1-LDCHK)2000,1240,1240
 1240 ENDCYCL=1.0
      LDCHK=0
      SOVRHD=0.0
      GO TO 2000
C  BASIC EXCHANGE PACKAGE
 2000 I = IQUE(NEXT)
      SWPOVD = 0.000300
      IF (STAT(I,2)) 2001,2001,2010
C IF GR 0 CHANNEL I NEEDS LOADING
 2010 CONTINUE
      CLOCK = CLOCK + EXOVO
      SOVD = SOVD + EXOVO
```

```
      MSIZE = GENR(I,6)
      IF(MSIZE + NDRUM - MAXDRM) 2011,2011,2012
C  NO LOAD
 2012 IF(MDRUM(I,3)) 2016,2013,2013
 2013 CONTINUE
      MDRUM(I,3) = -1
      SAVE(I,1) = CLOCK
      WAITC = WAITC + 1.0
 2016 NOLOAD = NOLOAD + 1
      GO TO 300
C LOAD OK
 2011 CONTINUE
      IF(MDRUM(I,3)) 2014,2015,2015
 2014 WAITT = WAITT + ( CLOCK - SAVE(I,1))
 2015 MDRUM(I,3) = MSIZE
      NDRUM = NDRUM + MSIZE
      STAT(I,2) = -1.0
      STAT(I,7) = 2.0
      STAT(I,8)=GENR(I,6)
      CLOCK = CLOCK + EXOVF
      SOVD = SOVD + EXOVF
      GO TO 200
 2001 CONTINUE
      IF (STAT(I,6)) 2002,2002,2020
C IF   GR 0  CHANNEL I IS QUITING .
 2020 CONTINUE
      MSIZE = STAT(I,8)
      NDRUM = NDRUM - MSIZE
      CLOCK = CLOCK + EXOVO
      SOVD = SOVD + EXOVO
      DO 2021 IA = 1,9
 2021 STAT(I,IA) = -1.0
      GO TO 210
C NORMAL EXCHANGE
C EXCHANGE METHOD 1 NO CON
 2002 CONTINUE
      MSIZE=MDRUM(I,3)
      CORET = CORET + 1.0
```

```
      COREUT = COREUT + STAT(I,8)
      GO TO 2100
 2100 CONTINUE
      IF(MCORE(I,4)) 2151,2151,2150

C PROGRAM IS IN CORE

 2150 TLOAD = 0.0
      TDUMP = 0.0
      TEDUMP=0.0
      TELOAD= 0.0
      GO TO 2152
 2151 CONTINUE
      IF(LASTI) 2154,2154,2153
 2154 TDUMP = 0.0
      TEDUMP=0.0
      GO TO 2155

C EXCHANGE LAST USER

 2153 TDUMP = STAT(LASTI,8) * 0.000003
      TEDUMP = TDUMP
      STAT(LASTI,7) = 2.0
      MCORE(LASTI,4) = -1
      MDRUM(LASTI,4) = 1
 2155 TLOAD = STAT(I,8) * 0.000003
      TELOAD =  TLOAD
      STAT(I,7) = 1.0
      MCORE(I,4) = 1
      MDRUM(I,4) = -1

C ADVANCE CLOCK CORRESPONDING TO EXCHANGE TIMES

2152  TEXCH = TEXCH + TDUMP + TLOAD + SWPOVD
      TEFEXCH = TEFEXCH + TELOAD + TEDUMP + SWPOVD
      SOVD = SOVD + SWPOVD
      SWPOVD = 0.0
      CLOCK=CLOCK + TLOAD + TDUMP
      LASTI = I
      GO TO  220

C     OUTPUT

  500 CONTINUE
      IF(WAITC) 559,559,560
 559  WAITC = 1.0
  560 CONTINUE
```

```
      AWAIT = WAITT/ WAITC
      COREF = COREUT/ CORET
      EXEFF = (TEXCH - TEFEXCH)/TEXCH
      ASWOVD=SOVD/CYCNT
      ASWAP=TEXCH/CYCNT
      DO 551 I=1,10
  551 DATA(I) = DIST(I)/DST
   11 TOTOUT(IR,1)=VARPARA
      TOTOUT(IR,2)=CYCNT
      TOTOUT(IR,3)=CYTMAVE
      TOTOUT(IR,4)=AVNQUE
      TOTOUT(IR,5)=QAVE
      TOTOUT(IR,6)=AOVRHD
      TOTOUT(IR,7)=COMPEFF
      TOTOUT(IR,8)=AOVLD
      TOTOUT(IR,9 )=QMAX
      TOTOUT(IR,10)=CYTIMMX
      TOTOUT(IR,11)=FMNQUE
      TOTOUT(IR,12)=SOVRHDM
      TOTOUT(IR,13)=FMNOVLD
      TOTOUT(IR,14)=NU
      TOTOUT(IR,15)  =  TEXCH
      TOTOUT(IR,16)  =  TEFEXCH
      TOTOUT(IR,17)  =  EXEFF
      TOTOUT(IR,18)  =  COREF
      TOTOUT(IR,19)  =  COREUT
      TOTOUT(IR,20)  =  CORET
      TOTOUT(IR,21)  =  AWAIT
      TOTOUT(IR,22)  =  NOLOAD
      TOTOUT(IR,23)  =  DST
      TOTOUT(IR,24)  =  SOVD
      TOTOUT(IR,25)  =  AVERAGE(1,6)
      TOTOUT(IR,26)  =  NU
      TOTOUT(IR,27)=ASWOVD
      TOTOUT(IR,28)=ASWAP
      TOTOUT(IR,29)=REQUEST
      DO 571 IM=1,10
      IN = IM + 30
      TOTOUT(IR,IN) = DATA(IM)
 571  CONTINUE
      IR=IR+1
570   CONTINUE
      IF(ISTOPF )10,10,558
  558 CONTINUE
      IR=IR-1
      PRINT 51
      PRINT 60
```

```
      PRINT 88
      PRINT 89
      PRINT 54
      PRINT 55
      PRINT 80
      PRINT 81
      PRINT 82
      PRINT 515,(((TOTOUT(I,J),J=1,7),(TOTOUT(I,J),J=27,28)),I=1,IR)
      PRINT 52
      PRINT 83
      PRINT 84
      PRINT 85
      PRINT 516,(((TOTOUT(I,J),J=8,14),TOTOUT(I,29)),I=1,IR)
      PRINT 51
      PRINT 90, ((TOTOUT(IT,IS),IS=15,20),IT=1,IR)
      PRINT 91, ((TOTOUT(IT,IS),IS=21,26),IT=1,IR)
      PRINT 92,((TOTOUT(IT,IS),IS=31,40),IT=1,IR)
      STOP
      END
C     UNIFORM AND GAUSSIAN R. N. GENERATOR
      SUBROUTINE RANDOM
      DIMENSION RNU(12)
      COMMON GENR,AVERAGE,DIST,JN,JOBTYPE,CLOCK,DST,IOUTCON
      COMMON MAXDRM,MAXDSC,MAXCOR,IR
      COMMON GAUSRN, UNIRN
      ARN = UNIRN
      FIV = 1953125.*390625.
      DO 20 I=1,12
      IF(ARN)11,10,11
 10   ARN=0.490843
 11   Z=ARN*FIV
      Z1=Z*1.0E-10
      NZ=Z1
      FLOAT=NZ
      RNU(I)=Z1-FLOAT
 20   ARN=RNU(I)
      GETRN=0.0
      DO 30 I=1,12
      CALC=RNU(I)+GETRN
 30   GETRN=CALC
      GAUSRN=CALC/6.0
      UNIRN=RNU(12)
      RETURN
      END
```

```
C      JOB GENERATOR
       SUBROUTINE SET(JX)
       DIMENSION UNUM(6)
       DIMENSION GENR(50,10) , AVERAGE(10,8) , DIST( 10)
       COMMON GENR,AVERAGE,DIST,JN,JOBTYPE,CLOCK,DST,IOUTCON
       COMMON MAXDRM,MAXDSC,MAXCOR,IR
       COMMON GAUSRN, UNIRN
   51  FORMAT(1H1)
   60  FORMAT(////////)
   63  FORMAT(//44X,20HFIGURE 6 (CONTINUED)//)
  615  FORMAT(I4,F10.1,2X,2I6,5F12.4,I9)
       I = 0
       J = JX
       JN = JN + 1
       DO 600 IC=1,5
       CALL RANDOM
       UNUM(IC) = GAUSRN
  600  CONTINUE
       CALL RANDOM
       RN = UNIRN
       TEMP = 0.0
       DST = DST + 1.0
  612  I = I + 1
       TEMP = TEMP + AVERAGE(I,7)
       IF(RN - TEMP) 610,611,611
  610  DIST(I) = DIST(I) + 1.0
       GO TO 614
  611  IF(I-10) 612,614,614
  614  CONTINUE
C ARRIVAL TIME
       CALL RANDOM
       GENR(J,1) = CLOCK + LOGF(UNIRN)*(-AVERAGE(I,1))
C LOAD
       GENR(J,2) = AVERAGE(I,2)*UNUM(1)
C ACTIVE TIME
       GENR(J,3)=AVERAGE(I,3)*UNUM(2)
C I/O TIME
       GENR(J,4)=AVERAGE(I,4) * UNUM(3)
C REPEATS
       GENR(J,5)=AVERAGE(I,5)*UNUM(4)
C SIZE
       GENR(J,6)=AVERAGE(I,6)*UNUM(5)
  620  CONTINUE
       MSIZE = GENR(J,6)
       IF(MSIZE-MAXCOR) 618,619,619
  619  GENR(J,6) = MAXCOR - 100
```

[illegible]

```
      GO TO 620
  618 CONTINUE
      IF(IOUTCON-4)616,621,616
  621 IF(IR-1)617,622,616
  622 JP=JP+1
      IF(JP-31)617,623,623
  623 JC=JC+1
      IF(JC-41)624,625,625
  624 IF(JC-1)617,627,617
  627 PRINT 51
      PRINT 60
      PRINT 63
      GO TO 617
  625 IF(JD)617,626,617
  626 JD=1
      PRINT 51
      PRINT 60
      PRINT 63
      GO TO 617
 617  PRINT 615,JN,CLOCK,JX,I,(GENR(J,K),K=1,5),MSIZE
  616 CONTINUE
      RETURN
      END
      END
     300.0        1.0        1.0        1.0        1.0    8000.0        0.25
     300.0        2.0        2.0        2.0        2.0    8000.0        0.25
     300.0        3.0        3.0        3.0        3.0   16000.0        0.25
     300.0        4.0        4.0        4.0        4.0   16000.0        0.25
     900.0
     900.0
     900.0
     900.0
     900.0
     900.0
400000 32000 32000
     0.200      0.150      0.100      0.075      0.050      0.025      0.050
     0.010      0.002     3600.0      0.1
 2   120 2 4
400000 32000 32000
     0.200      0.150      0.100      0.075      0.050      0.025      0.050
     0.010      0.002     3600.0      0.1
 2   220 2 4
400000 32000 32000
     0.200      0.150      0.100      0.075      0.050      0.025      0.050
     0.010      0.002     3600.0      0.1
 2 1 320 2 4
```

APPENDIX III

STATUS PRESERVATION IN THE 1604-160 SATELLITE ENVIRONMENT

The current 1604-160 Satellite System Control Programs do not provide for executive directed program exchange during a job run. Rather all switching between job input queries is performed by the Monitor program during the interjob interval.

While program exchange as a technique will be of limited effectiveness until faster external storage devices are available (with present tapes, the exchange time would be 16 seconds), the status preservation technique necessary is of value. The following analysis indicates the nature of the status preservation problem associated with such an exchange and describes methods of resolution.

To provide the desired break-in capability, a priority approach should be followed. Upon receipt of a satellite request, the batch job would be halted, the environment preserved, and the program transferred. The remote user would then be serviced to completion and the batch job continued (Figure III-1). While the system provides little in the way of exchange techniques, it does allow a detailed investigation of the problems involved in environment preservation.

The batch job can be interrupted at any time except when a transfer to peripheral equipment is actually in progress. This problem was handled previously by the use of a flag that could be set by the 1604 and sensed by the 160. This procedure was described by Lt. R. Hogg and Lt. D.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

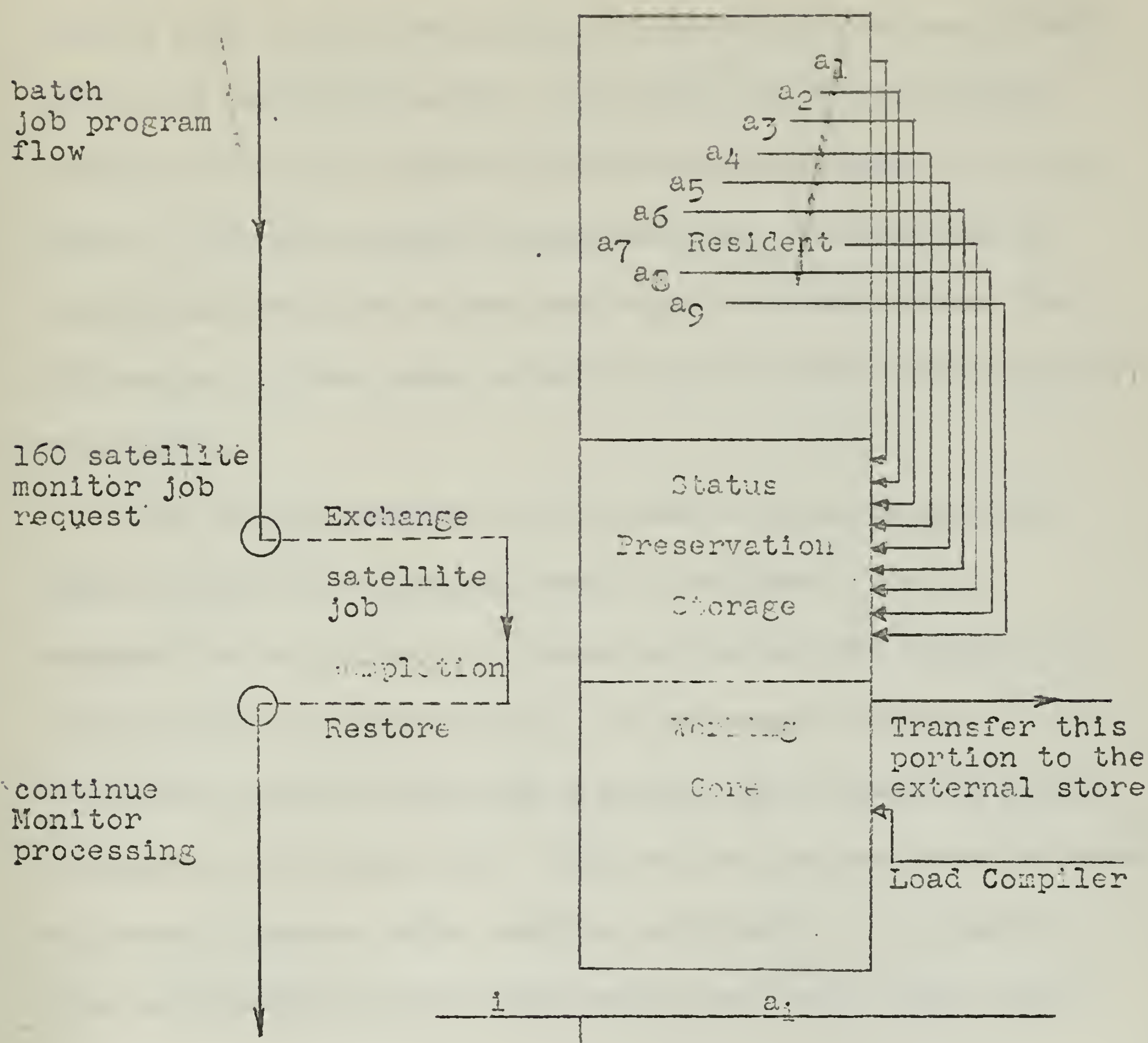
[illegible]

[illegible]

FIGURE III-1

Basic Program Flow

Operation of Preservation Cycle

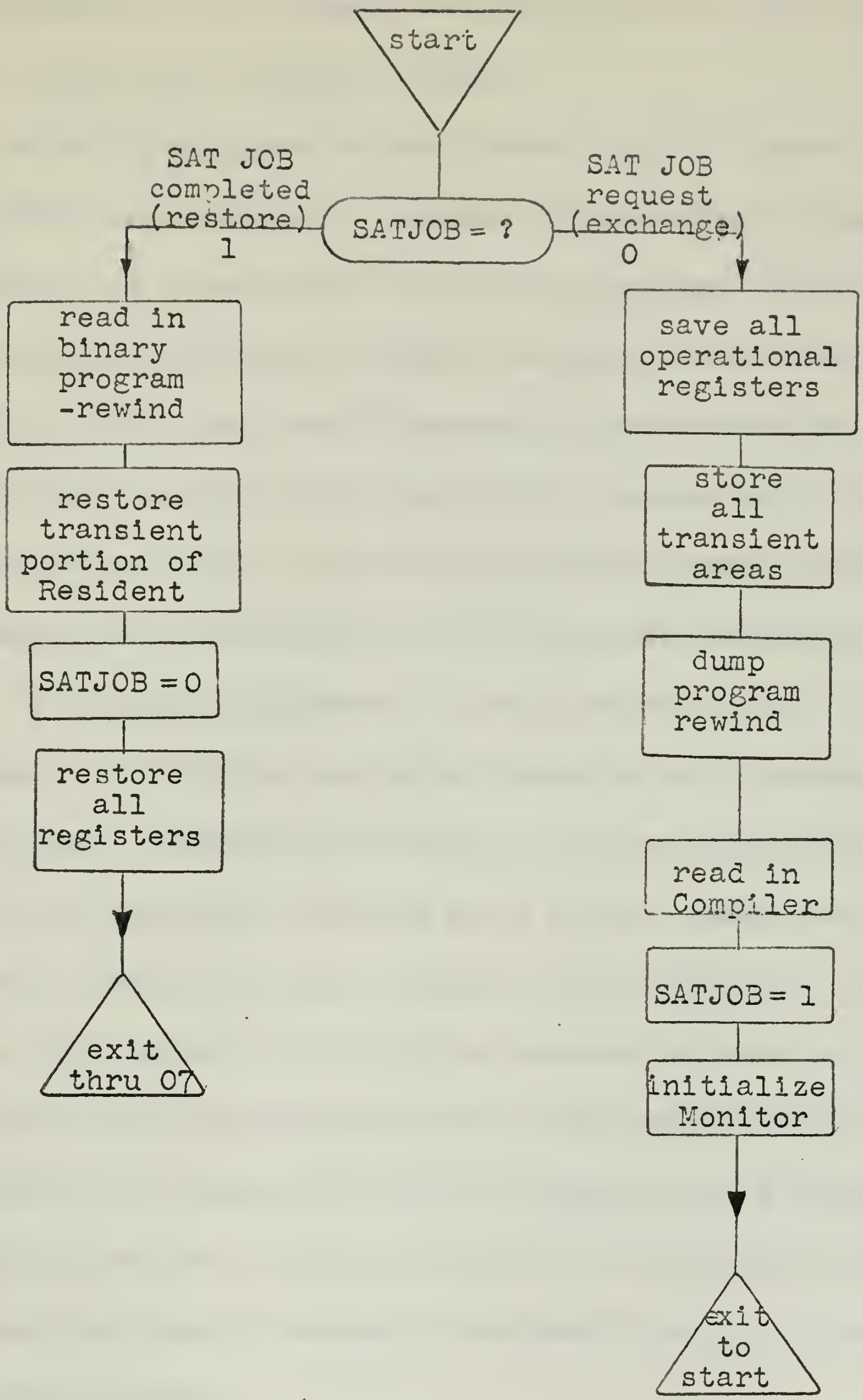


i	a_i
1	Cell 00 - Cell 07
2	Control Information and Monitor Tape Assignment Table
3	Read Buffer
4	Write Buffer
5	Listable Output Buffer
6	Punched Card Buffer
7	Stacked Job Buffer
8	Program Start Table
9	Monitor Control Cells

Glover in previous papers (14 and 15). Since then, an Interrupt Lockout select code has been provided in the hardware. It is recommended that this select code be used during all transfer operations and no wait loop will be required in the 160. The Lockout code will be selected immediately before the transfer and deactivated immediately upon completion. To preserve maximum system sensitivity this code must be selected and deselected on every pass through a repetitive loop. The 160 interrupt will then remain on the line until the 1604 is free to service the request.

Upon sensing a monitor type job satellite request interrupt the 1604 will branch to the Exchange portion of Resident. It is this Exchanger that will preserve the present environment and initialize the Monitor Routine for the satellite job. As discussed in Section 5.2. the environment could be stored either in internal tables or with the program transferred to the external store. With only one user ever being exchanged the storage in internal tables would be most practical. A storage area could be provided immediately after Resident with only a slight change in Resident programming.

The operation of the proposed Exchange routine is shown in the flow chart of Figure III-2. To permit the restoring of the interrupted job upon completion of the satellite job a SATJOB flag should be used. Upon sensing a job termination the Monitor routine would check the status of the SATJOB flag, if it were set, and hence a satellite job had just been completed, a branch to the Exchanger would be taken.



1604 SATELLITE SYSTEM
STATUS PRESERVATION ROUTINE

FIGURE III-2

If the flag was not set, a normal batch job termination would be assumed and the normal flow of Resident followed.

The double instruction per word format of the 1604 poses the only difficult status problem. At present, it is impossible to sense the status of the Interrupt Exit Flip Flop which provides the only indication of whether the interrupt occurred on an upper or lower instruction. Normally, programs exit from the Interrupt Processor through the Interrupt Entry Location (Cell 07) and control is automatically returned to the proper half word. In the proposed satellite system, however, a new program would be injected into the system with new initial conditions. If the absolute requirement of status preservation is to meet, the status of this flip flop must be both preserved and be capable of being restored. Programs can be written to determine this status but they are time consuming. Once the status has been determined it can be reset by artificially creating a divide overflow from the half word desired. This causes an interrupt to be generated and with the suitable flag checking in the Interrupt Processor, control can be passed to the Exchanger with the Interrupt Exit Flip Flop properly set. A hardware modification has been designed to both sense and reset this and its implementation should be seriously considered if this type of operation is to become common.

The transient portions of Resident can be stored in a 1K area of storage. While the following list is not complete it includes the main areas that must be preserved. These areas are also shown in Figure III-1.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

1. Cell 00 - Cell 07. These contain the I/0 transfer control words and the interrupt exit. If cell 07 is preserved along with the upper or lower status of the Interrupt Exit Flip Flop, restoring these will provide reentry to the point at which the program was interrupted

Due to the difference in Resident arrangements, the location of the following areas is not a fixed quantity. The approximate length is given after each area in octal notation.

2. Control Information and Monitor Tape Assignment Table (20)

3. Read Buffer (200)

4. Write Buffer (230)

5. Listable Output Buffer (20)

6. Punched Card Buffer (20)

7. Stacked Job Buffer (30)

8. Program Start Table (40)

9. Monitor Control Cells (10)

These areas should be saved and then initialized prior to starting a satellite job. Due to the fact that a batch job has probably been interrupted a different output tape should be designated for the satellite job. A careful determination should be made of the other areas that require initial values to be reset.

The actual saving and resetting should proceed in a logical fashion, preserving those quantities that will be changed by the Exchanger first. The tape to be used for the transfer can be either preset in the system

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

or brought in from the 160 as an input parameter.

Due to the nature of the FORTRAN compiler it would be advisable to dump the entire core above Resident for all Exchanges. To reduce transfer times, schemes to determine the portion of core actually in use should be investigated for use in later systems. As a time saving step, upon completion of any transfer the tape should be rewound immediately so that it is at the load point ready for the next transfer operation.

This exchange technique will provide an invaluable start towards a more sophisticated multiprogramming system. The question of environment preservation is a vital one and its solution will apply to any type of further system development. Availability of high speed transfer mediums and/or the use of space allocation will still require the perfect preservation methods developed through actual implementation of this proposal.